TURING 图灵程序设计丛书

PACKT PUBLISHING



[英] Ben Frain 著
王永强 译

Responsive Web Design with HTML5 and CSS3

响应式Web设计

HTML5和CSS3实战

- 前端设计人员必备教程
- 图文并茂，轻松掌握最新设计技术
- 全面应用HTML5和CSS3，一步跨入最前沿

人民邮电出版社
POSTS & TELECOM PRESS



Responsive Web Design with HTML5 and CSS3

响应式Web设计

HTML5和CSS3实战

[英] Ben Frain 著
王永强 译

人民邮电出版社
北京

图书在版编目（ＣＩＰ）数据

响应式Web设计 : HTML5和CSS3实战 / (英) 弗雷恩 (Frain,B.) 著 ; 王永强译. -- 北京 : 人民邮电出版社, 2013.1
（图灵程序设计丛书）
书名原文: Responsive Web Design with HTML5 and CSS3
ISBN 978-7-115-29922-2

Ⅰ. ①响… Ⅱ. ①弗… ②王… Ⅲ. ①超文本标记语言—程序设计②网页制作工具 Ⅳ. ①TP312②TP393.092

中国版本图书馆CIP数据核字(2012)第266700号

内 容 提 要

移动互联网时代到来了。本书将当前 Web 设计中热门的响应式设计技术与 HTML5 和 CSS3 结合起来，为读者全面深入地讲解了针对各种屏幕大小设计和开发现代网站的各种技术。不仅讨论了媒体查询、流式布局、相对字体、响应式媒体，更将 HTML5 和 CSS3 的相关知识点一并讲解，是学习最新 Web 设计技术不可多得的佳作。

本书适合各个层次的 Web 开发和设计人员阅读。

图灵程序设计丛书
响应式Web设计：HTML5和CSS3实战

◆ 著　[英] Ben Frain
译　王永强
责任编辑　李松峰
◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷
◆ 开本：800×1000　1/16
印张：15.5
字数：360千字　2013年 1 月第 1 版
印数：1－4 000册　2013年 1 月北京第 1 次印刷
著作权合同登记号　图字：01-2012-5558号
ISBN 978-7-115-29922-2

定价：49.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

译　者　序

蒂姆·伯纳斯–李在 1991 年制作并发布了第一个网站，如今刚刚过去 21 个年头。在这 21 年里，计算机和互联网快速发展，这个世界的面貌也日新月异。在这个过程中，网页设计从无到有，从简单渐至专业，从可有可无变为广受关注。网页设计方法也在跟随时代不断创新，从最初简单的文字排版，到表格布局，再到 DIV+CSS，直到现在广为流行的网格布局、流式布局等，设计师和开发者们一直致力于为全球网民提供更好的设计观感和使用体验。

iOS 和 Android 的发布，掀起了移动互联网的浪潮，智能手机、平板电脑、智能家电等新设备层出不穷，我们的世界变得更加精彩纷呈。但这却给网页设计带来了新的挑战，在面对形形色色的终端设备、千差万别的屏幕分辨率，以及良莠不齐的网络连接质量时，目前的设计方法显得力不从心。我们无法预料用户的设备和网络状况，更不可能为每种设备专门设计一套网站，那么在移动互联网时代，如何创新为用户提供更好的设计和体验呢?

2010 年 5 月 25 日，伊桑·马科特发表在 A List Apart 上的一篇文章，为我们打开了思路。在这篇名为《Responsive Web Design》的文章中，伊桑·马科特将三种已有的技术整合在一起，提出了响应式网页设计的概念，用以解决我们当前遇到的设计难题。响应式网页设计概念一经提出就大受欢迎（当然也有争议），很多设计师和开发者纷纷实践并完善这种理念。两年多的时间里，涌现出了越来越多的响应式网站，针对响应式设计的工具和资源也日渐丰富。截至目前，响应式设计是使用一套代码为各类设备提供良好设计效果和使用体验的最佳设计方法。

你肯定对响应式设计有所耳闻，也可能看过一些响应式设计的技巧或相关文章，但它们都零零散散不成体系，无法让你真正理解并掌握响应式设计，也无法指导你立即开始响应式设计的实践。这本书将会带你系统地学习响应式网页设计的方法论，书中涵盖了响应式设计的思想、方法、工具、技巧、HTML5、CSS3、相关资源，以及针对老版本浏览器的兼容方案等内容，并以实际案例循序渐进地讲解了如何创建一个优雅高效易于维护的响应式网站。希望这些正是你所需要的。

“得功有法，秉一应万”，这是太极拳修炼的目标，我想也可以作为响应式设计的目标。修炼好响应式设计这门功夫，能够让你安然自在，以一应万。元芳，潜心修炼吧！

感谢裕波、杨海玲老师及朱巍老师，让我有机会翻译本书。非常感谢图灵各位编辑的辛勤工作，尤其感谢李松峰老师的细心指导。另外要感谢图灵社区，我在这里受益匪浅。最后，感谢我的老婆，本书的翻译离不开她的支持、监督与参谋。

我在翻译本书时尽力保证信与达，雅则不敢奢望。翻译中的错误在所难免，欢迎广大读者指正。如果对本书有任何意见、建议或想法，请发送邮件至 wyqbailey@gmail.com 或反馈至图灵社区。

王永强

2012 年 10 月　成都

前　言

如果你想给自己的网站做一个单独的“手机版”，请三思而后行！响应式网页设计提供了一种设计方法，可以使同一网站在智能手机、桌面电脑，以及介于这两者之间的任意设备上完美显示。这种方法能够根据用户的屏幕尺寸，合理地为现有及将来的各种设备提供最佳的浏览体验。

本书提供了一整套方法，用来将一个现有的固定宽度的网站设计变成响应式的。此外，本书应用 HTML5 和 CSS3 提供的最新最有用的技术，扩展了响应式网页设计的方法论，以便网站更简洁、更易于维护。本书还讲解了编写和发布代码、图片、文件的最佳实践。

只要你懂 HTML 和 CSS，你就能制作响应式网站。

本书内容

第 1 章，HTML5、CSS3 及响应式设计入门，定义了什么是响应式网页设计，展示了一些响应式设计的网站示例，重点强调了使用 HTML5 和 CSS3 的优势。

第 2 章，媒体查询：支持不同的视口，讲解了什么是媒体查询，如何实现媒体查询，以及如何针对设备能力匹配 CSS 样式，将其应用于任意设计。

第 3 章，拥抱流式布局，讲解了流式布局的优点，以及如何将一个现有的固定宽度设计轻松地转换为流式布局，怎样使用 CSS 框架快速搭建响应式网页。

第 4 章，响应式设计中的 HTML5，探讨了使用 HTML5 技术的诸多好处，比如更简洁的代码、语义化标签、离线存储，以及无障碍网页应用辅助技术。

第 5 章，CSS3：选择器、字体和颜色模式，展示了 CSS3 选择器的强大威力，可以让你轻松地指定和改变任何元素。还讲解了通过@font-face 声明来使用漂亮的网络字体，另外讲解了新的 CSS3 颜色模式如 RGB(A)和 HSL(A)。

第 6 章，用 CSS3 创造令人惊艳的美，展示了如何使用纯粹的 CSS3 代码实现文字阴影、盒阴影和渐变效果。还涵盖了如何使用多重背景图片，以及如何通过字体文件创建图标。

第 7 章，CSS3 的过渡、变形和动画，讲解如何仅使用 CSS3 来创建和转换屏幕上的元素，

并制作动画效果。

第 8 章，用 HTML5 和 CSS3 征服表单，阐述了在所有设备上（从最新的智能手机到桌面版浏览器）都能良好运行的跨浏览器表单开发技巧。

第 9 章，解决跨浏览器问题，讲解了如何保证老版本的 Internet Explorer 可响应，如何将一组链接修改成移动设备上的一个菜单，如何为高分辨率显示器提供不同内容，以及如何使用 Modernizr 框架分条件地加载资源文件。

准备工作

你必须对 HTML 和 CSS 很熟悉。有一点 JavaScript 基础会很有帮助。良好的电影品味也很有益处。

本书读者

你是否正在开发两套网站，一套给移动设备，一套给大显示器？又或者你已听说过“响应式网页设计”但却不确定如何将 HTML5、CSS3 和响应式设计融合在一起？如果是，那么本书可以让你在所有竞争对手之前，将你的网站提升到一个更高层次。

本书面向那些正在使用 HTML 4.01 和 CSS 2.1 开发固定宽度网站的网页设计师和开发人员，讲解了如何使用 HTML5 和 CSS3 制作可适应任意屏幕尺寸设备的响应式网站。

本书约定

本书中使用几种不同的文字样式来区分不同类型的信息，具体约定如下。

正文中的代码使用如下格式：

“HTML5 接受宽松语法，例如 `<sCRipt SrC=js/jquery-1.6.2.js></script>` 这样的语句也是有效的。”

代码段使用如下格式：

```
<div class="header">
  <div class="navigation">
    <ul class="nav-list">
      <li><a href="#" title="Home">Home</a></li>
      <li><a href="#" title="About">About</a></li>
    </ul>
  </div> <!—end of navigation -->
</div> <!—end of header -->
```

当要专门强调代码块中的某一部分时，则对相关行或条目使用粗体格式：[编者注：网页

格式无法实现此效果。]

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 96%; /* 最外围的DIV */
}

#header {
  margin-right: 10px;
  margin-left: 10px;
  width: 97.9166667%; /* 940 ÷ 960 */
}
```

新术语，以及**重要词汇**也使用粗体。文中提到的那些菜单、对话框中的文字，会使用粗体或大写来标注，如：“导航按钮不再做背景色切换，内容区的 THESE SHOULD HAVE WON 按钮和侧边栏的**详细信息**按钮消失了，而字体也与设计文档相差甚远。”

> 这个图标表示警告或重要提醒。

> 这个图标表示提示或技巧。

读者反馈

我们时刻欢迎你的反馈，以便了解你对本书的看法。你的宝贵意见有助于我们提升书籍的质量。

一般的阅读反馈，可直接发送电子邮件至 feedback@packtpub.com，请在邮件标题中注明书名。

如果你在某个领域内有专长且有兴趣编写相关书籍，请访问 www.packtpub.com/authors 查看作者指南。

客户支持

现在你已是 Packt 图书的尊贵读者了，我们有一系列的售后支持，保证你的消费物有所值。

错误

尽管我们已经对书籍作了仔细校对以保证内容准确，但错误在所难免。如果在书中发现

任何的文字或代码错误，非常欢迎你将这些错误提交给我们，这样可以帮助我们在后续版本中改正错误，避免其他读者产生不必要的误解。一旦发现错误，请登录 http://www.packtpub.com/support，选择书名，点击 errata submission form（提交勘误）链接，然后填写具体的错误信息即可。只要你提交的勘误通过验证，勘误信息就会上传到我们的网站，或者追加到已有勘误列表中，显示在该书的勘误页面[1]。

盗版

对所有媒体来说，互联网盗版都是一个棘手的问题。Packt 很重视版权保护。如果你在互联网上发现我们公司出版物的任何非法复制品，请及时告知我们相关网址或网站名称，以便我们采取补救措施。

如果发现可疑盗版材料，请通过 copyright@packtpub.com 联系我们。对您帮助我们保护作者权益、确保我们持续提供高品质图书的行为表示敬意。

问题

如果你对本书有任何问题，请联系 questions@packtpub.com，我们会尽力解决。

① 对本书中文版的勘误，请访问 www.ituring.com.cn/bood/1055 提交。——编者注

致　　谢

首先要感谢网络社区。没有大家集思广益的才智、慷慨大方的文档和开放共享的解决方案，我不可能在互联网行业做出现在这点还稍稍令我自豪的事情。

其次，我要感谢响应式网页设计之父 Ethan Marcotte。他和我从未谋面或交谈过，但他的方法论现在每时每刻都在影响着我制作网页的方法。毫无疑问，假如有哪些地方造成读者对响应式方法论的误解，那都要怪我自己表达不好。

最后，感谢我的家人。看过（同样被低估的）《义海倾情》[①]的人都知道："血浓于水。其他人都是陌生人。"

①《义海倾情》Wyatt Earp：http://movie.douban.com/subject/1293055/）。——译者注（以下若非特别注明，均为译者注。）

目　　录

HTML5、CSS3 及响应式设计入门

1

直到最近，网站设计普遍还在使用固定宽度（如 960 像素），以期给所有终端用户带来较为一致的浏览体验。这种固定宽度设计在笔记本电脑上显示刚刚好，而在高分辨率显示器上却会在两边多出些空白。

但现在有了智能手机。苹果公司的 iPhone 第一次带给我们真正意义上易用的手机上网体验，之后其他公司纷纷效仿。现在人们可以舒服地使用手机上网冲浪，不用再像过去那样需要有“挑圆片”[①]世界冠军一样的灵活拇指，才能在小屏幕上看看网页。此外，消费者在家中上网时优先使用小屏幕设备（如平板电脑、上网本）正成为趋势。一个不争的事实是，使用小屏幕设备上网的人数正在以前所未有的速度增长。与此同时，27 英寸和 30 英寸的大显示器也在快速普及。上网设备之间的尺寸差距与日俱增。

幸运的是，面对不断发展的浏览器和上网设备，我们有可行的解决方案。采用 HTML5 和 CSS3 技术的响应式网页设计，可以使网站兼容多种设备和屏幕。而这种方法的最佳之处，在于不需要什么服务器端方案，也完全可以实现。

本章内容

- 支持小屏幕设备的重要性
- 什么是移动网站设计
- 什么是响应式网页设计
- 优秀响应式网页实例赏析
- 视口和屏幕的区别
- 安装和使用修改视口的浏览器扩展程序
- 使用 HTML5 编写更简洁的标记
- 使用 CSS3 解决常见的设计问题

① 一种游戏，详见 http://en.wikipedia.org/wiki/Tiddlywinks。

1.1 为什么智能手机很重要（而老版的 IE 不再重要）

虽然统计数据一般仅用作粗略参考，但来自 gs.statcounter.com 的统计数据却值得注意。从 2010 年 7 月到 2011 年 7 月的 12 个月中，全球手机浏览器的使用量从 2.86%上升至 7.02%。Internet Explorer 6 的使用率则是从 8.79%下降到 3.42%。到 2011 年 7 月，Internet Explorer 7 的使用率也降到了 5.45% 。如果客户老是要求："我们的网站必须兼容 IE6 和 IE7！"你可以反驳说："我们应该把精力花在更有价值的地方。"用手机上网的人比用台式机和笔记本中的 IE6 或 IE7 上网的人多多了。你可以听到全球前端开发工程师震耳欲聋的欢呼声！

越来越多的人使用小屏幕设备上网，这些设备上的浏览器在设计时都考虑到了如何显示好现有网站。手机浏览器会将一个标准网页缩小至与设备可视区域（标准技术术语叫做"视口"）恰好匹配。然后用户在自己感兴趣的内容区域上放大浏览。这样看起来已经挺好了，那作为前端设计师和工程师的我们，为什么还要在这上面继续优化呢？



在 iPhone 或 Android 手机上浏览的网页越多（如上图所示的那样），就越能深刻体会到为什么我们还需要继续优化。为了看清楚页面内容，需要不停地放大、缩小页面，然后为了看到视口外的文字，再左、右拖动，结果一不小心点了一个链接，你说讨厌不讨厌？我们当然可以做得更好！

1.2 响应式设计一定是最佳选择吗

如果预算充足且形势需要，做一个真正的“手机版”网站是首选。这样就可以基于用户的设备、位置、连接速度，以及包括技术特性在内的其他变量来提供不同的内容、设计和交互。举一个实际的例子，假设有一家比萨连锁店，它有一个标准版的网站和一个手机版网站，手机版网站可以基于增强现实功能、用户当前 GPS 位置找到附近的比萨店。单独一个响应式设计远远不能支撑这种解决方案。

然而，不是所有项目都要求如此完美。大多数情况下，根据视口大小为用户提供与之匹配的视觉效果还是优先选择。例如，针对大多数网站，虽然从服务器端请求的内容相同，但我们可以根据不同设备改变网页的显示方式。在小屏幕设备上，可能是将次要内容放在主体内容之下，或者最坏情况下将其直接隐藏；也可能是将导航按钮改造成便于手指操作，而不是只提供一些需要精确瞄准才能点击的玩意儿！字体大小也应该恰到好处，便于阅读，不再需要不停地在屏幕上拖来拖去。同样，在迎合小屏幕的同时，我们也不想降低笔记本和台式机用户的浏览体验。既然我们意在兼容万物，那给那些配备 1900 像素甚至更大屏幕的用户提供一点额外改进又有何不可呢？简而言之，我们需要那些能响应各种设备大小的完美设计。

1.3 响应式网页设计的定义

响应式网页设计（RWD，Responsive Web Design）这个术语，由伊桑·马科特（Ethan Marcotte）提出。他在 A List Apart 发表了一篇开创性的文章，将三种已有的开发技巧（弹性网格布局、弹性图片、媒体和媒体查询）整合起来，并命名为响应式网页设计。这个术语还有一堆表示相同意思的其他叫法，如流式设计、弹性布局、塑料布局、流体设计、自适应布局、跨设备设计以及弹性设计。

上面仅列举了其中一部分！不过，正如马科特等人所说，真正的响应式设计方法不仅仅只是根据视口大小改变网页布局。相反，它是要从整体上颠覆我们当前设计网页的方法。以往我们先是针对桌面电脑进行固定宽度设计，然后将其缩小并针对小屏幕进行内容重排；现在我们应该首先针对小屏幕进行设计，然后逐步增强针对大屏幕的设计和内容。

一句话概括响应式网页设计

如果要用一句话概括响应式网页设计，我觉得它是针对任意设备对网页内容进行完美布局的一种显示机制。相反，如果需要根据不同设备提供特定的内容和功能，那就需要一个真正的“手机版”网站。这种情况下，手机版网站会提供与桌面版网站完全不同的用户体验。

1.4　为什么要在响应式设计上停滞不前

响应式网页设计能够根据视口变化控制页面文档流，但我们可以走得更远。HTML5 提供了比 HMTL 4 更多且更加语义化的标签。CSS3 的媒体查询是响应式设计不可或缺的组成部分，CSS3 的其他功能给了我们前所未有的灵活性。我们将挣脱背景图像和复杂的 JavaScript 代码的羁绊，拥抱简洁的 CSS3 渐变、投影、字体、动画和变换。

在使用 HTML5 和 CSS3 制作响应式网页之前，让我们先来欣赏一些值得学习的实例，看看哪些高手正在使用新潮的响应式 HTML5 和 CSS3 绝技创造奇迹，而我们可以从他们的开创性杰作中学到些什么。

1.5　响应式网页设计示例

若要全方位测试你自己或别人的响应式网站，则需要针对每一种设备和不同的屏幕尺寸，分别准备不同的测试系统。尽管这是最完美的办法，但通过改变浏览器窗口大小其实就可以完成大多数测试。除此之外，还有很多第三方插件和浏览器扩展可供选择，通过它们可以将当前浏览器窗口或视口设定为指定像素。必要时，还可以自动将当前浏览器窗口或视口切换成为默认大小（如 1024×768 像素）。这样你就可以轻松地测试不同屏幕视口尺寸下的网站效果。

> **迷恋像素？忘了它吧！**
>
> 进入了响应式网页设计的教堂，就不要再迷恋像素（px）这个度量单位，因为大多数情况下我们不会用像素，而会使用相对度量单位（em 或百分比）。相对单位更方便我们审查其他响应式设计作品，查看设计的变更之处。

1.5.1　下载视口调试工具

Internet Explorer 用户请下载安装 Microsoft Internet Explorer Developer Toolbar，下载地址如下：http://www.microsoft.com/download/en/details.aspx?id=18359

如果你在使用 Safari，虽然 ResizeMe（http://web.me.com/aaronholla/Safari_Extensions/ResizeMe.html）的功能类似且免费，但我最爱 Resize（http://resizeSafari.com）。

Firefox 用户请下载 Firesizer（https://addons.mozilla.org/en-US/firefox/addon/firesizer/），Chrome 请下载 Windows Resizer（https://chrome.google.com/webstore/detail/kkelicaakdan-hinjdeammmilcgefonfh）。

不喜欢使用浏览器扩展？还有一个方法：我写了个简单 HTML 页面来显示浏览器窗口的

当前视口高度和宽度。页面用了 jQuery 框架，获取当前的视口的高度和宽度并显示出来。你可以在浏览器新标签页中打开这个页面，调整窗口大小，然后切回你要测试的页面查看效果。这个超级简单的“What size is my viewport page?”页面地址如下：http://benfrain.com/easily-display-the-viewport-size-of-your-page-for-responsive-designs/

> **视口和屏幕尺寸**
>
> 视口和屏幕尺寸不是同一个概念。**视口**是指浏览器窗口内的内容区域，不包含工具栏、标签栏等。也就是网页实际显示的区域。屏幕尺寸指的是设备的物理显示区域。需要注意的是有些浏览器调整工具显示的尺寸包含浏览器的其他元素，诸如地址栏、标签栏和搜索栏，而有些工具则不是这样。在下面的截图中，实际的视口尺寸显示在右上角（1156×921 像素），同时 Firesizer 插件将窗口尺寸显示在右下角（1171×1023 像素）。



现在，我们带着所有需要的工具，开始鉴赏最好的响应式网站。启动你钟爱的浏览器，打开浏览器调整工具，访问 http://thinkvitamin.com/ 。

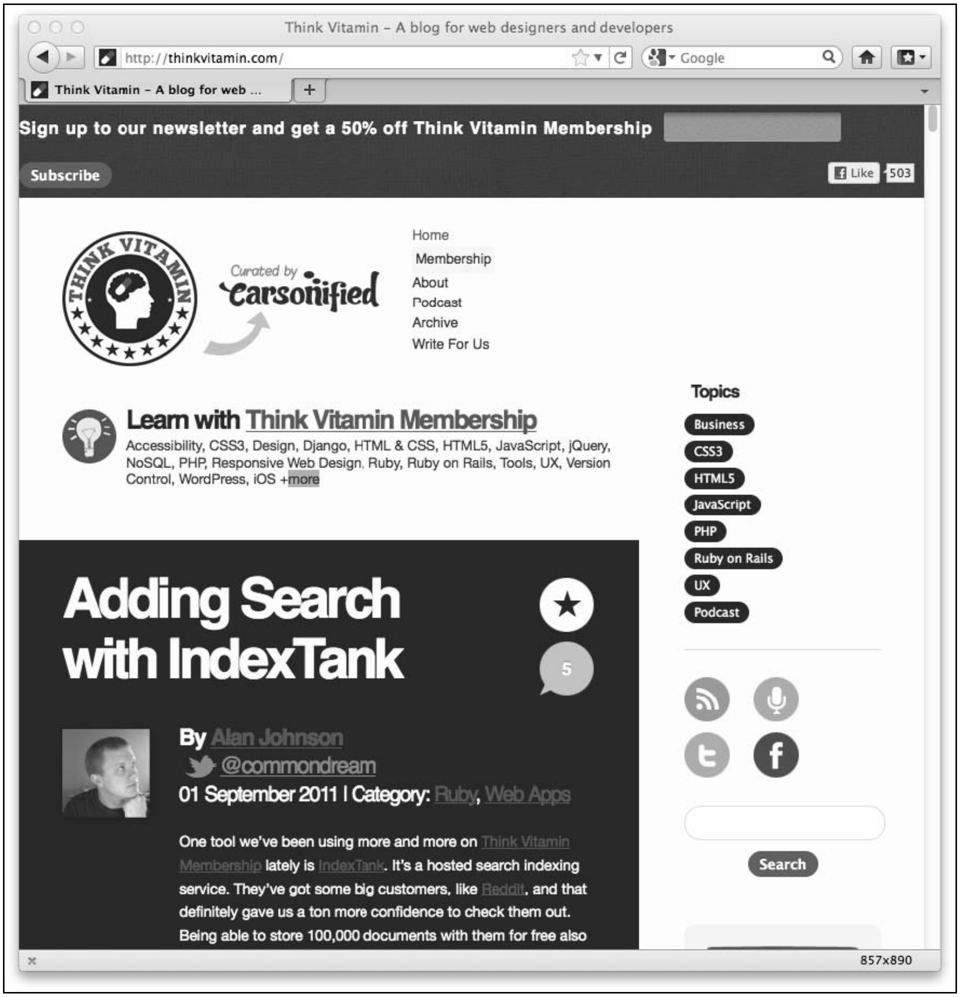
如果你的视口宽度大于 1060 像素，你会看到如下图所示的布局：



如果你的视口宽度介于 930 像素到 1060 像素之间，你会看到如下图所示布局：

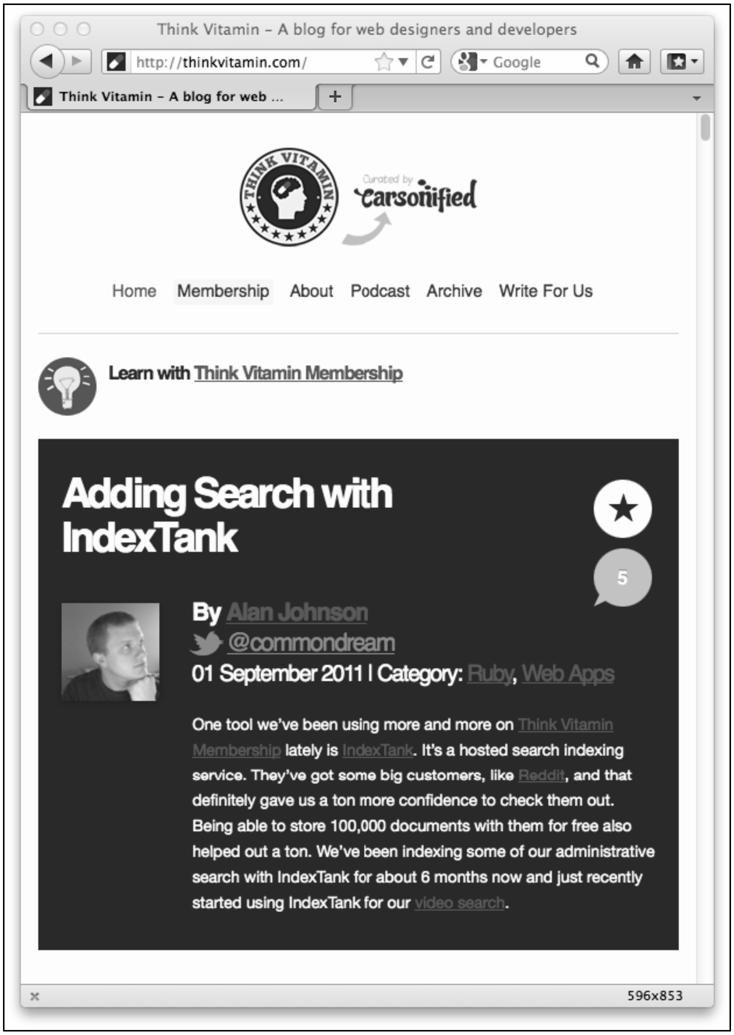


注意看 logo 旁边的主导航菜单是如何变化的。主内容区右侧的图标一个挨着一个排列，界面上的一切都合理可用，最重要的是，没有一样从屏幕上溢出。现在让我们再看看视口宽度小于 880 像素的效果，截图如下：



头部的效果基本没变，但注意右侧的侧边栏还是变窄了一些。图标排成了 2 行 2 列，同时文本块做了适当调整，其中的文本流相应地发生变化。

不过，将视口宽度减小到小于 600 像素，你就会发现一个重大的变化，如下图所示：



怎么样？整个侧边栏对我们的新视口做出了响应，网站最重要的内容区占据了整个浏览器窗口宽度。注意看头部的导航链接现在是水平排列的，而不是被放在 logo 的旁边。另外，logo 本身的大小也做了调整。以上所有的这些变化有助于根据视口尺寸为用户创建更好的体验。

让我们来看看另一个例子：http://2011.dconstruct.org/。视口较宽时（大于 1350 像素）网站效果如下图所示：



请特别注意那 9 张图的排列格式。当你减小视口宽度时（小于 960 像素），注意看界面发生了什么变化。三行三列的图片排列方式变成了三行两列外加下方一行三列，如下图所示：



继续减小视口宽度，在小于 720 像素的时候我们会遇到另一个设计“断点”。头部导航链接转换成了带图片的导航区域，这为触屏导航提供了更好的操作区域：



继续减小视口宽度，当小于 480 像素时，图片排列方式又变成了第一行 2 张图片，第二行 3 张，第三行 4 张。这些图片的大小在视口宽度缩小至大约 300 像素时又发生了变化。下面的截图显示了其在 iPhone 上的效果：



1.5.2　在线创意源泉

推荐一个响应式设计创意收集网站：http://mediaqueri.es。虽然这里收集的网站并不是全部都采纳完整的响应式方法论，即先针对小视口进行设计，然后逐步针对大视口进行渐进增强支持。但就目前来看，响应式设计方法兴起的时间不长，再考虑到响应式网页设计的各种可能性，这里确实有很多能让我们汲取创意的范例。尽管调整视口大小来浏览网站能说明响应式网页设计的理念，但这没有展示出 HTML5 的优势。HTML5 的优势事实上在幕后发挥。所以让我们将注意力转移到幕后，来了解一下 HTML5 的优秀之处。

1.6　为什么 HTML5 很优秀

在创建可以通过 W3C 标准验证的页面时，HTML5 强调简化标签，仅链接那些我们必须的 CSS、JavaScript 和图片文件。智能手机用户只能使用有限的带宽访问我们的页面，而响应式设计的一个主要目标就是，网站不仅要对用户有限的视口做出响应，还要以最快的时间加载网页。虽然移除冗余的标签元素只能节省一点字节，但积少成多！

较之上一个版本的 HTML（HTML 4.0.1），HTML5 进行了极大的改进，提供了许多新特性。前端开发工程师可能主要对 HTML5 的新语义元素感兴趣，这些新元素给搜索引擎提供了含义更加丰富的代码。HTML5 也可以在表单提交等基本网页交互场景中对用户做出反馈，一般不再需要复杂的 JavaScript 表单处理流程。同样，这为我们的响应式设计来带来的好处就是，允许我们创建更加简洁和快速的代码。

1.6.1　省时省力

HTML 文档的第一行都是使用 Doctype（文档类型声明）开头。老实说，这段代码一般都是由代码编辑器自动生成或者是从模板文件中粘贴过来的（没人会真的手工敲出完整的 HTML 4.01 文档类型声明吧？）。标准的 HTML 4.01 网页的文档类型声明如下所示：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
    TR/html4/loose.dtd">
```

如今在 HTML5 中，则化繁为简：

```
<!DOCTYPE html>
```

如前所述，我写页面的时候从来不会手工敲出文档类型声明，我猜你也不会。——那你嚷嚷半天有什么实际意义？ 不要着急，还得往页面中链接 JavaScript 或 CSS 文件呢。在 HTML 4.01 中，链接一个脚本文件的正确方法如下：

```
<script src="js/jquery-1.6.2.js" type="text/javascript"></script>
```

HTML5 中变得更简单：

```
<script src="js/jquery-1.6.2.js"></script>
```

如你所见，`type` 属性不再是必需的。链接 CSS 文件与此类似。HTML5 也接受非常松散的语法。如，`<sCRipt SrC=js/jquery-1.6.2.js></script>`和之前所举例子一样有效，尽管我们省略了脚本源文件地址两边的引号，而且标签和属性名大小写字母混用，但 HTML5 并不介意，这样的代码照样能够通过 W3C 的 HTML5 验证工具（http://validator.w3.org/）的验证。如果你在写代码时容易丢三落四，这或许是个好消息。如果你想从代码中剔除每一个可能多余的字符，这点尤其有用。在代码编写方面还有很多细节可以让我们的工作更加轻松从容，但我猜你一定期待 HTML5 更多的精彩内容。那就让我们来快速浏览一下 HTML5 新增的语义元素。

1.6.2 新增了语义化标签元素

制作标准的 HTML 页面时，头部和导航一般都是标配，如下所示：

```
<div class="header">
  <div class="navigation">
    <ul class="nav-list">
      <li><a href="#" title="Home">Home</a></li>
      <li><a href="#" title="About">About</a></li>
    </ul>
  </div> <!—end of navigation -->
</div> <!—end of header -->
```

看看我们用 HTML5 如何实现：

```
<header>
  <nav>
      <ul id="nav-list">
        <li><a href="#" title="Home">Home</a></li>
        <li><a href="#" title="About">About</a></li>
      </ul>
   </nav>
</header>
```

怎么样？对于表示每个结构化元素的毫无个性的`<div>`标签（虽然为应用样式加了 class 名），HTML5 为我们提供了一些有语义的元素。页面中的一些通用结构体如头部和导航（后面还会看到更多）有了独立的标签。使用`<nav>`标签会使我们的代码变得更有语义，如同告诉浏览器，“嘿，这块内容就是导航”。这对我们来说是件好事，但更重要的是对搜索引擎来也是件好事。搜索引擎能比以前更好地理解我们的网页，并相应地评定网页内容。

在编写 HTML 页面时，我经常提醒自己，这些页面在最终发布到互联网上之前，将按流程交给负责后台开发的同事（你懂的，就是那些使用 PHP、Ruby、.NET、ColdFusion 等等语言的帅哥们）。为了和后台开发人员保持良好的协作关系，我通常都会给代码中的`</div>`结束标签添加注释，以确保其他人（也包括我自己）能够轻松地确定`<div>`元素在哪里结束。HTML5 可以让我们省去大部分这样的工作。查看 HTML5 代码时，假如看到一个`</header>`结束标签，你会立即明白哪个标签结束了，不用注释。

以上我们只窥见了 HTML5 语义增强的一斑。先别激动，我们还要认识一位朋友。如果没有它，就没有网页设计的新时代，它就是 CSS3。

1.7 CSS3 为响应式设计和更多创新奠定了基础

如果你从 20 世纪 90 年代中期就开始从事网页设计，应该记得当时所有的设计都基于表格布局。而且，样式与内容是交织在一起的。CSS（层叠样式表）作为一种将设计与内容

分离的方法被引入。网页设计师们花了一些时间才走进基于 CSS 设计的美丽新世界，诸如“CSS 禅意花园”（http://www.csszengarden.com）这样的网站从视觉上形象地展示了基于 CSS 设计的网站所能达到的效果，为我们铺平了前进的道路。从那以后，CSS 成为了定义网页表现层的标准方法，CSS 2.1 是当前正式批准的 CSS 标准。CSS3 尚未被正式批准，但这并不意味着它的大部分内容现在无法使用。W3C 工作组的说明如下：

> CSS3 是在 CSS 2 基础上按模块构建的，以 CSS 2.1 标准为核心。每个模块都会增加功能或是替换 CSS 2.1 标准中已有章节。CSS 工作组的设想是新的 CSS 模块不会与 CSS 2.1 标准冲突，这些模块只会追加新功能，改进已有规定。

大部分 W3C 标准草案读起来（不可避免地）像法律条文。简单来说，CSS3 是一种附加模块式构造，而不是大一统的标准。由于 CSS3 以 CSS 2.1 为核心，所以 CSS 2.1 的功能照样可以使用。不仅如此，某些相对成熟的 CSS3 模块（并不是所有模块都得到了相同程度的支持）今天也可以较为广泛地使用了，因此不必等待整个规范得到批准。

1.7.1　底线：CSS3 不破坏任何东西

关于 CSS3，最鼓舞人心的一点应该是在老版本浏览器中使用它们无法解析的属性，不会造成任何问题。它们（包括 Internet Explorer 6、7、8）会欣然忽略那些无法解析的属性。这使我们能够为那些先进的浏览器进行渐进增强设计，同时为老版本浏览器提供合理的降级处理。

1.7.2　CSS3 如何解决日常设计问题

我们来看一个在大多数项目中都会遇到的设计问题——给界面元素创建圆角效果，可能是设计选项卡式界面，也可能是给诸如头部这样的块状元素应用圆角。在 CSS 2.1 中可以使用滑动门技术（http://www.alistapart.com/articles/slidingdoors/）来实现，即将一张背景图片放在另一张后面。对应的 HTML 代码比较简单：

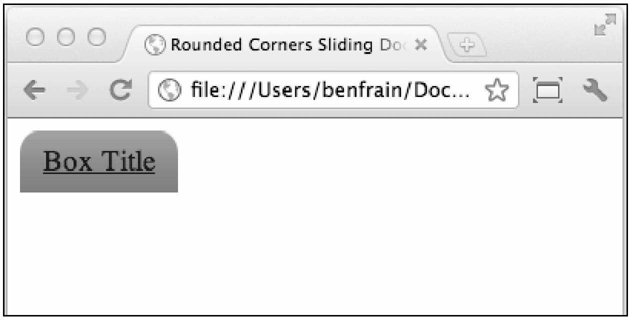
```
<a href="#"><span>Box Title</span></a>
```

为了给`<a>`元素添加圆角背景，我们需要制作两张图片。第一张叫做 `headerLeft.png`，15 像素宽 40 像素高；第二张叫做 `headerRight.png`，宽度要超过头部可能设定的最大宽度（本例中宽度为 280 像素）。两张图片合起来组成滑动门。当元素宽度增加时（`<span>`标签内的文字增加），背景图片会填满背景空间，这样就形成了一个普遍适用的圆角解决方案。下面是本例中的 CSS 代码：

```
a {
  display: block;
  height: 40px;
  float: left;
  font-size: 1.2em;
```

```
  padding-right: 0.8em;
  background: url(images/headerRight.png) no-repeat scroll top right;
}
a span {
  background: url(images/headerLeft.png) no-repeat;
  display: block;
  line-height: 40px;
  padding-left: 0.8em;
}
```

Google Chrome 浏览器（v16）中的效果如下：



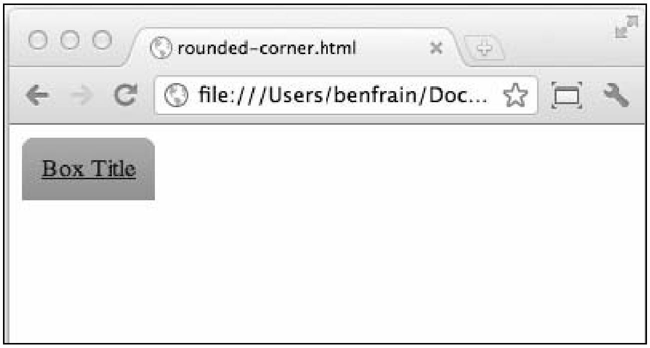
上面的方法解决了设计问题，但却需要增加一个多余的标签（从语义上看<span>标签没有实际意义）和两次额外的服务器端 HTTP 请求（两张图片）来创建屏幕上的视觉效果。除此之外，我们还可以使用 CSS“雪碧”（Sprite）技术，将两张图片合成一张，然后使用 `background-position` 属性来调整定位，虽然这样还可以节省一点带宽，但仍然不是一个灵活的方案。如果客户要求圆角更大一点怎么办？或者要求修改颜色怎么办？我们必须得重新修改图片。悲哀的是，作为前端设计师和工程师的我们，在 CSS3 出现以前，就身处这样的窘境中。女士们先生们，我已经看到了光明的未来，那是由 CSS3 塑造的未来！我们将上述的 HTML 代码简化成这样：

```
<a href="#">Box Title</a>
```

CSS 代码改为如下所示：

```
a {
  float: left;
  height: 40px;
  line-height: 40px;
  padding-left: 0.8em;
  padding-right: 0.8em;
  border-top-left-radius: 8px;
  border-top-right-radius: 8px;
  background-image: url(images/headerTiny.png);
  background-repeat: repeat-x;
}
```

下面的效果图展示了在同样的浏览器（Chrome v16）中呈现的 CSS3 版的圆角按钮效果：



在上面的代码中，先前的两张图片已被替换为一张沿 *x* 轴平铺的 1 像素宽图片。虽然图片只有 1 像素宽，但高度仍然是 40 像素，比任何可能出现在其中的元素都高。在使用图片作为元素背景时，一定要对其高度“高度注意”，以预防内容溢出。不幸的是，这样会产生更大的图片，需要更多带宽。尽管如此，和先前完全使用图片的解决方案不同的是，CSS3 提供了 `border-radius` 及其相关属性来帮助我们设置圆角。客户想让圆角更平滑一点，比如改成 12 像素？没问题，只需要将 `border-radius` 的属性值改为 `12px`，搞定！CSS3 的圆角属性简单、灵活，并且 Safari（v3+）、Firefox（v1+）、Opera（v10.5+）、Chrome（v3+）和 Internet Explorer 9（及 IE 10）都支持它。微软对 IE 9 能够支持这个属性得意洋洋（我希望你能感受到我在此处所表达的一丝讽刺），他们甚至专门设计了一个交互页面来演示使用 `border-radius` 属性能达到的各种效果。演示页面的地址如下：

http://ie.microsoft.com/testdrive/html5/borderradius/default.html

CSS3 可以更进一步，不再需要渐变背景图片，而是使用浏览器渲染的效果。浏览器对这个特性的支持不是很好，但按照 `linear-gradient(yellow, blue)`这个基本思路，任意元素的背景都可以用 CSS3 生成的渐变来渲染。

渐变可以设定为纯色，即传统的 HEX 颜色值（如`#BFBFBF`）；也可以使用任何一种 CSS3 颜色模式（更多详情请见第 5 章）。如果你想给老版本浏览器的用户设置替换渐变的纯色背景（否则他们看不到任何背景），如下所示的 CSS 代码可以给不支持渐变的浏览器提供一个纯色背景：

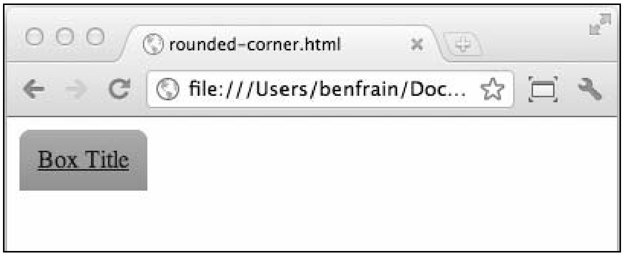
```
background-color: #42c264;
background-image: -webkit-linear-gradient(#4fec50, #42c264);
background-image: -moz-linear-gradient(#4fec50, #42c264);
background-image: -o-linear-gradient(#4fec50, #42c264);
background-image: -ms-linear-gradient(#4fec50, #42c264);
background-image: -chrome-linear-gradient(#4fec50, #42c264);
background-image: linear-gradient(#4fec50, #42c264);
```

`linear-gradient` 属性会指示浏览器从第一个颜色值（即例子中的`#4fec50`）开始，渐变至第二个颜色值（`#42c264`）。

你可能注意到了，CSS 代码中的 `background-image:linear-gradient` 属性使用不同前缀（例如`-webkit-`）重复了多次。这种写法允许不同的浏览器（包括`-moz-`代表的 Mozilla Firefox，`-ms-`代表的 Microsoft Internet Explorer 等）厂商在发布正式版本之前之前，试验各自对 CSS3 新特性的实现，正式版本发布后就不再需要前缀。遵循样式表的层叠特性，我们将无前缀的代码放在最后，这样如果该特性可用，则会覆盖之前的声明。

1.8 看吶，不用图片

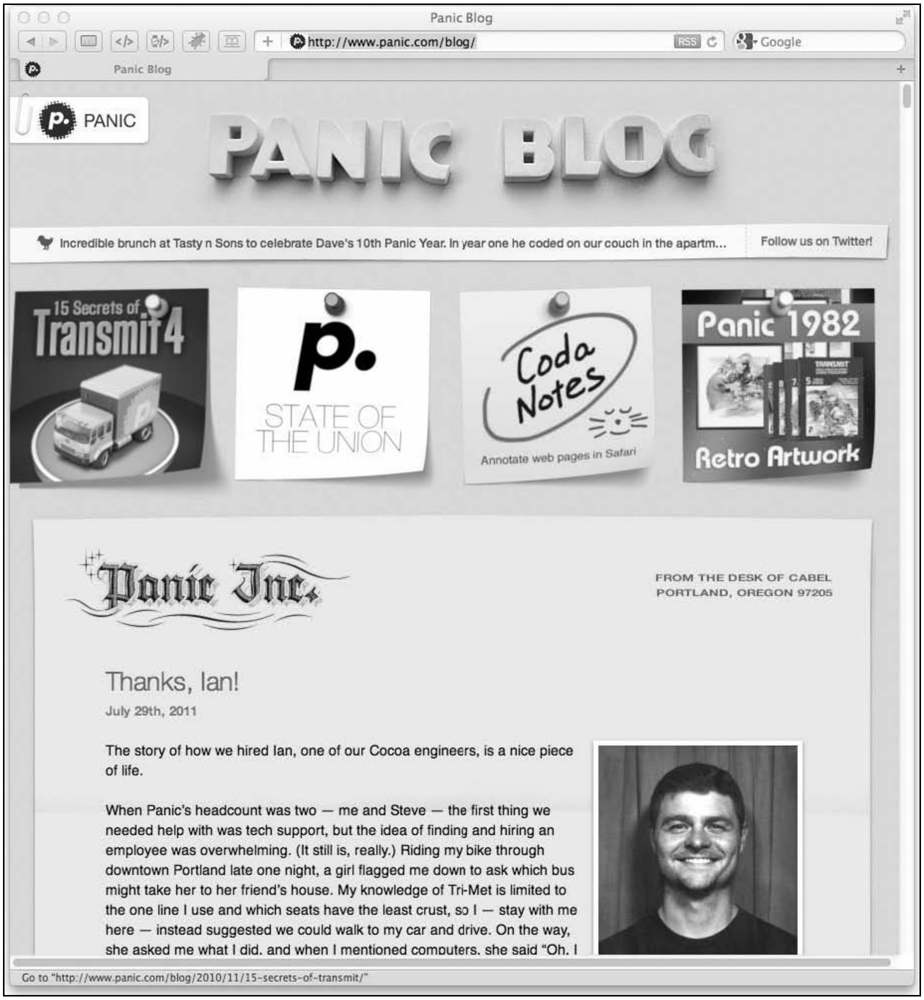
下面的截图展示了在同样的浏览器中完全使用 CSS3 制作的按钮效果:



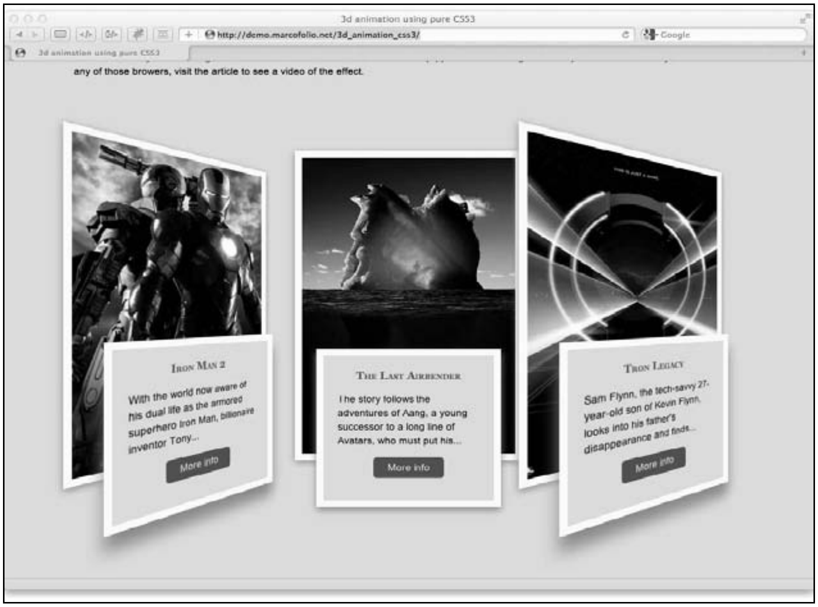
图片版和纯 CSS 版之间的视觉差异微不足道，我想这点你应该同意。相较于使用图片，使用 CSS3 制作视觉效果能使我们的响应式设计更加简洁。而且，现代浏览器都能很好地支持渐变，唯一需要权衡是那些对渐变支持不怎么到位的浏览器，如 IE 9 以及更低版本的 IE。

CSS3 还带来了什么

刚才，我们欣赏了一个 CSS3 能帮助我们完成的日常工作的小例子。接下来，我们想尝尝开胃大菜，看看 CSS3 能带给我们什么美味。启动 Safari 或 Chrome，访问 http://www.panic.com/blog/。遗憾的是这个网页不是响应式设计，我们关心的区域是页面上方的那一排记事贴。把鼠标悬停在上面就可以看到它们浮起来。很帅吧？过去，这种增强效果一般都要靠笨重的 Flash 或者 JavaScript 实现。而现在则可以完全通过 CSS3 的变换来实现。使用 CSS3 制作动画比 JavaScript 或 Flash 更轻量级，更好维护，因此对响应式设计来说很理想。支持该特性的浏览器会显示效果，不支持的则装傻跳过，仅将其看做一张静态图片而已。



另一个优秀的 CSS3 动画范例是 http://demo.marcofolio.net/3d_animation_css3。这个网站也不是响应式设计,但我们只关心其中用到的 CSS 技巧。先在 Internet Explorer 9 或者 Firefox 中看看（9.0 版本以前，Firefox 还不支持该 CSS3 模块），然后在 Safari 5+或者 Chrome 16+中看看效果。下面的截图效果未尽其意，如果你不准备亲自欣赏，那你得相信我的话——确实很赞：



酷炫的效果不仅仅只是 Webkit 核心的 Safari 和 Chrome 浏览器的专利。下面的例子在 Firefox 中也很炫，而且也是纯 CSS3 实现的：http://designlovr.com/ examples/dynamic_stack_of_index_cards/。



显然，这些效果不是每个网站都必需的。它们是“渐进增强”的完美印证。在不支持特殊效果的浏览器里，用户只会看到静态的图片。但使用现代浏览器的用户则可以享受增强的视觉效果。虽然支持 CSS3 3D 变换的浏览器非常有限，但支持诸如文字阴影、渐变、圆角、RGBA 颜色和多重背景图片等 CSS3 效果的浏览器则非常多，这为解决常年来令我们挠头的设计问题提供了灵活的方法。

1.9　HTML5 和 CSS3 现在就能用吗

任何工具或技术都应该只在应用程序需要它的时候使用。做为前端工程师/设计师，我们的项目往往因为财务预算而受到时间和资源上的限制。

Internet Explorer 7 和 8 都不支持 HTML5 新的语义元素，也不支持 CSS3 的新属性。如果一个网站的绝大多数用户都使用 Internet Explorer 7 或 8，那花费精力将其做成基于 HTML5 和 CSS3 的响应式设计没有太大意义。但这并不意味着我们做不到。第 9 章在讨论跨浏览器响应问题时，会介绍一些刚刚涌现的新工具（因为它们是用于修补旧浏览器缺陷的，所以也被称为 polyfill），用来修补那些不支持现代浏览器特性的老旧浏览器（主要是老旧的 IE），但最好是从一开始就采取切实可行的方法来实现响应式设计。

以我的经验，一开始通常会问自己以下问题。

- 客户是否想支持互联网用户增长最迅猛的市场？如果想，那响应式方法就很适合。
- 客户是否想要最简洁、快速，且易于维护的代码？如果想，那响应式方法就很适合。
- 客户能否理解用户体验可以且本应该根据浏览器不同而不同？如果可以理解，那响应式方法就很适合。
- 客户是否要求设计效果在所有浏览器中都保持一致，包括 IE 8 以及更低版本？如果是，响应式设计就不适合。
- 该网站的当前或预期客户中，是否有百分之七十以上的人可能使用 Internet Explorer 8 或者更低版本？如果是，则响应式设计不适合。

再次重申，在预算允许的情况下，一个完全定制的“移动”版网站比响应式设计更适合。澄清一下，我将那些完全专注移动平台、为移动设备用户提供不同内容/体验的解决方案称之为“移动网站”。提倡响应式设计方法的人，不会都认为响应式设计在任何情况下都可以替代“移动网站”。

1.10　响应式网页设计不是灵丹妙药

虽然有点“爹爹给婆婆拜年——多此一举”，但还是有必要再重申一遍：使用 HTML5 和 CSS3 的响应式网页设计不是解决所有设计和内容服务问题的灵丹妙药。和以往的网页设

计一样，项目的具体情况（即预算、目标用户以及网站用途）决定了其实现方式。根据我的经验，如果预算有限或开发一个完全定制的“移动网站”不太可行，那么响应式网页设计较之标准的固定宽度设计，总能提供更好的和无歧视的用户体验。

1.11 引导客户：网站不必在所有浏览器中表现一致

着手响应式设计之前，要跨越的最后一道障碍往往是思维定式。而且在某些情况下，这或许是最难克服的问题。例如，我经常被要求将已有的平面设计转换成使用 HTML/CSS 和 jQuery 构造的标准网页。依我的经验，平面设计师在创作设计图时，通常只会考虑固定宽度“桌面版”网站，考虑得更多的设计师真是罕有（我说罕有，意思是从来没见过）。接着我的职责就是在所有浏览器中以像素级别还原该设计。这项任务的最终成败取决于客户以及平面设计师的眼光。这种想法在有平面印刷设计背景的客户那里更是根深蒂固。他们的想法很好理解：客户签字认同了设计图，然后将其交给你、我这样的前端设计师/工程师，之后我们投入时间来确保最终代码在所有主流浏览器中的显示效果与原始设计尽可能接近——客户所见就是客户所得。

如果你尝试过让一个现代网页设计在 Internet Explorer 6、7 中，和在 Safari、Firefox 或 Chrome 等现代浏览器中的表现完全一致，你就会知道什么叫“蜀道之难，难于上青天”了。我经常会用整个项目近百分之三十的时间/预算来修复这些烂浏览器的固有缺陷和不足。这些时间应该用来为越来越多的现代浏览器用户开发增强体验或优化代码，而不应该浪费在为越来越少的 Internet Explorer 用户修补代码，绞尽脑汁地生造出圆角、透明图片、正确对齐的表单元素等效果。

遗憾的是，解决这个问题的唯一方法就是说服引导。客户需要明白为什么应该支持响应式设计，响应式设计的结果如何，为何最终设计不会也不应在所有视口和浏览器中表现一致。有些客户能理解，有些则不能。悲剧的是，有些客户仍然坚持要求在 Internet Explorer 6 中也要有一模一样的圆角和投影效果。

当我接手一个新项目时，不论响应式设计是否适合，我都会试着给客户说明以下几点。

- 允许页面显示效果在老旧浏览器中有细微的差别，这样可以使代码更易维护，将来更新的成本也更低。
- 让页面元素在那些老旧浏览器（如 Internet Explorer 8 及更低版本）中表现一致会导致网站增加大量的图片。这会使网站变慢，制作成本变高，而且更难维护。
- 现代浏览器可以理解的简洁代码等同于更快速的网站。快速响应的网站在搜索引擎中的评级高于慢腾腾的网站。
- 使用老旧浏览器的用户越来越少，使用现代浏览器的用户越来越多——我们应该支持大多数！

- ❑ 最重要的一点，支持现代浏览器，你就能尽情地享受响应式网页设计，它能响应不同设备的不同浏览器视口。

1.12　小结

本章，我们介绍了什么是响应式设计，并且欣赏了已有的优秀响应式设计范例。这些设计都是使用我们将在本书中讲解的工具和技术构建的。我们也已认同，以桌面电脑为中心的设计思想，应该转变成为未知设备而设计的思想。首先为最小的可视区域设计版式，然后在此基础上渐进增强用户体验。通过了解新的 HTML5 规范，我们确定 HTML5 的大部分内容可用，而且能够发挥其优势。换句话说，新的语义标签允许我们使用更简洁的代码创建比以往更具语义的网页。

实现响应式设计的关键技术是 CSS3。在使用 CSS3 添加渐变、圆角、文字阴影和动画等视觉效果之前，首先要让它来扮演一个更重要的角色，那就是利用 CSS3 的媒体查询，针对特定的视口设置特定的 CSS 规则。下一章我们将开始“响应式网页设计”的探索之旅。

媒体查询：支持不同的视口 2

如上一章所述，CSS3是由很多附加模块组合而成的。媒体查询就是其中的一个模块。媒体查询可以让我们根据设备显示器的特性为其设定CSS样式。例如，我们仅使用几行代码，就可以根据诸如视口宽度、屏幕比例、设备方向（横向或纵向）等特性来改变页面内容的显示方式。

本章内容

- 理解为什么响应式设计需要媒体查询
- 如何构造CSS3媒体查询
- 我们能够检测哪些设备特性
- 编写第一个CSS3媒体查询
- 为特定视口设定CSS样式
- 如何在iOS和Android设备上使用媒体查询

2.1 现在就能使用媒体查询

媒体查询已经被广泛使用，而且也被浏览器广泛支持（如Firefox 3.6+、Safari 4+、Chrome 4+、Opera 9.5+、iOS Safari 3.2+、Opera Mobile 10+、Android 2.1+和Internet Explorer 9+）。此外，要对老版本浏览器如Internet Explorer 6、7和8实现兼容修复（虽然基于JavaScript）也很容易。如果你现在想对Internet Explorer 6、7和8进行兼容修复，可以查阅第9章关于解决跨浏览器响应问题的内容。总之，没有什么理由能阻碍我们现在使用媒体查询。

> W3C对规范有一套完整的审批流程（如果你有空，可以去看看该流程的官方说明，地址：http://www.w3.org/2005/10/Process-20051014/tr），从工作草案（Working Draft，WD），到候选推荐标准（Candidate Recommendation，CR），再到提议推荐标准（Proposed Recommendation，PR），几年之后，才能成为W3C推荐标准（REC）。所以那些相对成熟的模块使用起来比较安全。比如，CSS3变换模块

> Level 3（http://www.w3.org/TR/css3-3d-transforms/）从 2009 年 3 月起一直处于工作草案状态，相对于候选推荐标准的媒体查询模块，浏览器支持远远不足。

2.2 为什么响应式设计需要媒体查询

没有 CSS3 的媒体查询模块，就不能针对设备特性（如视口宽度）设置特定的 CSS 样式。如果你仔细研读 W3C 关于 CSS3 媒体查询模块的规范，就会看到媒体查询的官方解释：

> HTML 4 和 CSS 2 目前支持为不同的媒体类型设定专有的样式表。比如，一个页面在屏幕上显示时使用无衬线字体，而在打印时则使用衬线字体。`screen` 和 `print` 是两种已定义的媒体类型。媒体查询让样式表有更强的针对性，扩展了媒体类型的功能。
>
> 媒体查询由媒体类型和一个或多个检测媒体特性的条件表达式组成。媒体查询中可用于检测的媒体特性有 `width`、`height` 和 `color`（等）。使用媒体查询，可以在不改变页面内容的情况下，为特定的一些输出设备定制显示效果。

2.2.1 媒体查询语法

CSS 媒体查询到底长什么样，更重要的是，它是怎么起作用的？

将下面这段代码插入到任意某个 CSS 文件的最后，然后预览与之关联的网页：

```
body {
  background-color: grey;
}
@media screen and (max-width: 960px) {
  body {
    background-color: red;
  }
}
@media screen and (max-width: 768px) {
  body {
    background-color: orange;
  }
}
@media screen and (max-width: 550px) {
  body {
    background-color: yellow;
  }
}
@media screen and (max-width: 320px) {
  body {
    background-color: green;
  }
}
```

在现代浏览器（如果是 IE，至少要 IE9）中浏览该网页并不断调整浏览器窗口宽度。页面的背景颜色就会根据当前的视口尺寸而发生变化。为了清晰起见，我在这里使用了颜色名称，但实际上最好使用十六进制颜色值，如#ffffff。

接下来，让我们继续分析媒体查询，学习如何对其进行充分利用。

2

如果经常使用 CSS 2 样式表，你就知道可以通过<link>标签的 media 属性为样式表指定设备类型（如显示屏或打印机）。具体说来，就是在 HTML 页面的<head>标签中插入一个如下面代码片段所示的 link 标签：

```
<link rel="stylesheet" type="text/css" media="screen" href="screen-styles.css">
```

媒体查询则能使我们根据设备的各种功能特性来设定相应的样式，而不仅仅只针对设备类型。可以将媒体查询想象成对浏览器的提问。如果浏览器回答“是”，则应用样式；如果回答是“否”，则不应用样式。相对于在 CSS 2 中能且只能问浏览器“你是一块显示屏吗？”，媒体查询能问的问题要多一点。例如，媒体查询可以问：“你是一块纵向放置的显示屏吗？”我们看看对应的实际代码：

```
<link rel="stylesheet" media="screen and (orientation: portrait)" href="portrait-
    screen.css" />
```

首先，媒体查询表达式询问了媒体类型（你是一块显示屏吗？），然后询问了媒体特性（显示屏是纵向放置的吗？）。任何纵向放置的显示屏设备都会加载 portrait-screen.css 样式表，其他设备则会忽略该文件。在媒体查询的开头追加 not 则会颠倒该查询的逻辑。例如，下面的代码就会颠倒前例中的效果，会使非纵向放置的显示屏设备加载样式文件：

```
<link rel="stylesheet" media="not screen and (orientation: portrait)" href="portrait-
    screen.css" />
```

也可以将多个表达式组合在一起。如，我们扩展一下前面的例子，限制只有视口宽度大于 800 像素的显示屏设备才能加载文件。

```
<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width:
    800px)" href="800wide-portrait-screen.css" />
```

更进一步，还可以写一个媒体查询列表。查询列表中的任意一个查询为真，则加载文件。全部查询都不为真，则不加载。例子如下：

```
<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width:
    800px), projection" href="800wide-portrait-screen.css" />
```

这里有两点需要注意。第一，媒体查询之间使用逗号分隔。第二，你会注意到在 projection 之后，没有 and，也没有任何特性/值的组合。没有后续表达式，意味着只要是 projection 就满足条件。本例中，样式会应用于所有的投影仪。

和以前编写 CSS 规则一样，基于媒体查询也可以按条件加载样式。在上面的例子中，我们在向页面的<head></head>标签中链接 CSS 文件时使用了媒体查询。除此之外，我们

还可以在 CSS 样式表中使用媒体查询。例如，将下面的代码插入样式表，在屏幕宽度小于等于 400 像素的设备上，h1 元素的文字颜色就会变成绿色。

```
@media screen and (max-device-width: 400px) {
  h1 { color: green }
}
```

还可以使用 CSS 的@import 指令在当前样式表中按条件引入其他样式表。例如下面的代码会给视口最大宽度为 360 像素的显示屏设备加载一个名为 phone.css 的样式表。

```
@import url("phone.css") screen and (max-width:360px);
```

切记，使用 CSS 的@import 方式会增加 HTTP 请求（这会影响加载速度），所以请谨慎使用该方法。

2.2.2　媒体查询能检测那些特性

创建媒体查询时，最常用的是设备的视口宽度（width）和屏幕宽度（device-width）。依我的经验，很少需要检测其他特性。但是，为方便查阅，下面列出了所有可供媒体查询检测的特性，希望其中有能让你心动的特性。

- width：视口宽度。
- height：视口高度。
- device-width：渲染表面的宽度（对我们来说，就是设备屏幕的宽度）。
- device-height：渲染表面的高度（对我们来说，就是设备屏幕的高度）。
- orientation：检查设备处于横向还是纵向。
- aspect-ratio：基于视口宽度和高度的宽高比。一个 16∶9 比例的显示屏可以这样定义 aspect-ratio: 16/9。
- device-aspect-ratio：和 aspect-ratio 类似，基于设备渲染平面宽度和高度的宽高比。
- color：每种颜色的位数。例如 min-color: 16 会检测设备是否拥有 16 位颜色。
- color-index：设备的颜色索引表中的颜色数。值必须是非负整数。
- monochrome：检测单色帧缓冲区中每像素所使用的位数。值必须是非负整数，如 monochrome: 2。
- resolution：用来检测屏幕或打印机的分辨率，如 min-resolution: 300dpi。还可以接受每厘米像素点数的度量值，如 min-resolution: 118dpcm。
- scan：电视机的扫描方式，值可设为 progressive（逐行扫描）或 interlace（隔行扫描）。如 720p HD 电视（720p 的 p 即表明是逐行扫描）匹配 scan: progressive，而 1080i HD 电视（1080i 中的 i 表明是隔行扫描）匹配 scan: interlace。
- grid：用来检测输出设备是网格设备还是位图设备。

在上述所有特性中，除 scan 和 grid 之外，都可使用 min 和 max 前缀来创建一个查询范围。例如，分析如下所示的代码片段：

```
@import url("phone.css") screen and (min-width:200px) and (max-width:360px);
```

这里对 `width` 应用了 `min` 和 `max` 来设定查询范围。这样 `phone.css` 文件只会引入视口宽度介于 200 像素至 360 像素的显示屏设备。

2.2.3　用媒体查询改造我们的设计

毫无疑问，你肯定知道 CSS 代表层叠样式表。所谓层叠，就是指样式表中后面的样式会覆盖前面相同的样式（除非前面的样式具有更高的针对性）。因此我们可以在样式表的开头设置基本样式，以便适应所有设计，然后使用媒体查询来进一步重写相应的部分。例如，先针对大视口设计（用户大多数情况下使用鼠标），将导航链接设计成简单的文字链接；然后再针对较小的视口，使用媒体查询重写这部分样式，为拇指一族提供更大的点击区域。

2.2.4　加载媒体查询的最佳方法

现代浏览器虽然可以智能地忽略与自身不匹配的样式文件，但它却不一定不下载这些文件。因此，将不同媒体查询的样式保存到独立的文件中没有太大好处（个人喜好或为便于组织代码除外）。使用多个独立的文件会增加用于页面渲染的 HTTP 请求数量，从而导致页面加载变慢。

Respond.js（https://github.com/scottjehl/Respond）是为 Internet Explorer 8 及更低版本增加媒体查询支持的最快的 JavaScript 工具，但它目前无法解析 CSS 的`@import` 命令。因此，建议在已有的样式表中追加媒体查询样式。使用如下语法即可在已有样式表中加入媒体查询：

```
@media screen and (max-width: 768px) {/*样式*/}
```

2.3　我们的第一个响应式设计

不知道你此刻怎么想，但我热切地渴望开始设计一个响应式网页！我们已经理解了媒体查询的原理，那就让我们试着驾驭它们，看看它们在实际工作中如何发挥作用。而且我正好有用来测试的项目。下面请允许我说点题外话……

我喜欢电影。但是，我常常发现自己和别人的观点不同（或许这就是我日复一日独居一室写代码的原因之一），尤其是关于什么是好电影什么是烂电影。每当奥斯卡奖提名宣布的时候，我经常会有一种强烈的恶心反胃感。我总觉得各种类型的电影都应获得嘉奖。

我打算自己弄一个名叫 And the winner isn't 的小网站，网址是 http://www.andthewinnerisnt.com/。这里会褒奖那些本应获奖的电影，批评那些（不该获奖却）获奖的电影，这里还有视频剪辑、经典语录、电影海报，以及能证明我没错的在线调查（我知道没必要，但我喜欢）。

2.3.1　我们的设计是固定宽度的，不要惊讶

和我以前骂过的那些从不考虑不同视口的平面设计师一样，我开始也基于 960 像素的固定宽度网格制作一个界面原型。虽然理论上最好是从移动（小屏幕）设备的体验出发开始设计，然后渐进增强，但实际上要让每个人都理解这种思路的好处恐怕还需几年时间。在此之前，更多的可能是要将现有的桌面版网页设计改造成响应式的。既然未来几年我们可能都会这样做，那我们的设计还要从固定宽度着手。下面的截图展示了一个粗略的固定宽度的界面原型：



将这个设计分解，可以看到它有几个简单通用的结构：头部、导航、边栏、内容区和页脚。恐怕你每周都会构建一遍类似的结构吧。

第 4 章会告诉你为什么应该使用 HTML5 标签。不过此时我将略过这点，因为我们想急切地检验刚讲的媒体查询技巧。所以我们将使用旧的 HTML 4 标签来进行我们的第一次媒体查询试验。使用 HTML 4 标签编写的没有实际内容的基本页面结构如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>And the winner isn't</title>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>

<body>

<div id="wrapper">
  <!-- the header and navigation -->
  <div id="header">
    <div id="navigation">
      <ul>
        <li><a href="#">navigation1</a></li>
        <li><a href="#">navigation2</a></li>
      </ul>
    </div>
  </div>
  <!-- the sidebar -->
  <div id="sidebar">
    <p>here is the sidebar</p>
  </div>
  <!-- the content -->
  <div id="content">
    <p>here is the content</p>
  </div>
  <!-- the footer -->
  <div id="footer">
    <p>Here is the footer</p>
  </div>

</div>
</body>
</html>
```

用 Photoshop 打开设计文档，我们可以看到头部和页脚都是 940 像素宽（两边各有 10 像素外边距），而侧边栏和内容区分别占据了 220 像素和 700 像素，相应地两边也各自有 10 像素的外边距。



首先，我们要在 CSS 中给页面主要的结构模块（头部、导航、侧边栏、内容区和页脚）设置一下样式。在插入重置样式之后，页面的主体 CSS 代码应该如下所示：

```
#wrapper {
      margin-right: auto;
  margin-left: auto;
  width: 960px;
}

#header {
  margin-right: 10px;
  margin-left: 10px;
  width: 940px;
  background-color: #779307;
}

#navigation ul li {
  display: inline-block;
}

#sidebar {
  margin-right: 10px;
  margin-left: 10px;
  float: left;
  background-color: #fe9c00;
  width: 220px;
}

#content {
```

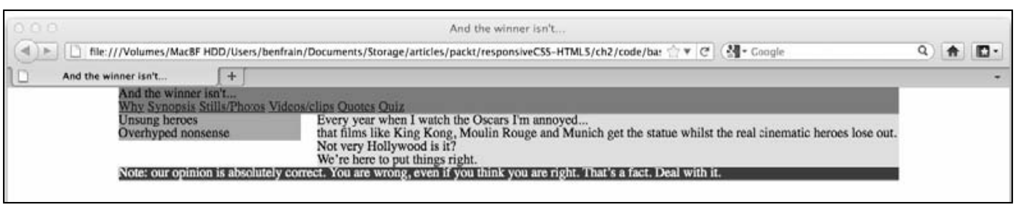
```
  margin-right: 10px;
  float: right;
  margin-left: 10px;
  width: 700px;
  background-color: #dedede;
}

#footer {
  margin-right: 10px;
  margin-left: 10px;
  clear: both;
  background-color: #663300;
  width: 940px;
}
```

为了说明页面的结构，我在给各个结构模块追加额外内容的同时，还为各个结构模块设置了不同的背景颜色。

> 重置样式就是一组 CSS 声明，用来覆盖不同浏览器渲染 HTML 元素时的各种默认样式。重置样式一般会被加入到主样式文件的开头，用来将各个浏览器的自有默认样式重置成统一表现，确保样式表中后续追加的样式在不同浏览器中有相同的显示效果。世界上没有完美的重置样式，许多开发者都有自己的一套。我在 HTML 4 页面中使用的重置样式，是在 Eric Meyer 的原版（http://meyerweb.com/eric/tools/css/reset/）基础上加上了我的一些个人偏好及技巧，这些技巧是我从另外一些天才如 Dan Cederholm（http://simplebits.com）的代码里学来的。如果你现在还没用过重置样式，那么强烈建议在你的 HTML 4 文档开头插入 Eric 的重置样式。我觉得针对 HTML5 文档有更好的选择，如 normalize.css（http://necolas.github.com/normalize.css/），第 4 章会详细介绍。

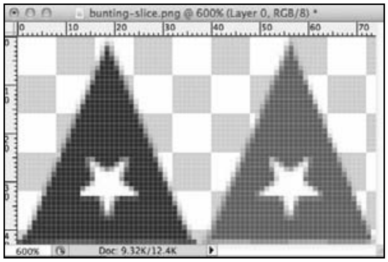
在一个视口大于 960 像素的浏览器中，页面的基本结构应如下所示：



还有好几种使用 CSS 制作同样效果的固定宽度左右分栏结构的方法，你肯定有自己偏爱的方法。不过这些方法都普遍存在一个问题，当视口缩减到小于 960 像素的时候，右侧的内容区就开始被截断。

2.3.2　响应式设计中要保证图片尽可能精简

为了说明上述代码结构的问题，我已经提前参照设计图在 CSS 中增加了一些装饰性样式。既然要做响应式设计，那我肯定会用最精简的办法来切取背景图片。例如，对于设计图顶部和底部的小彩旗，我不会制作一个长条图片，而只是切取其中的两面小旗。这个切片作为背景图在视口中水平重复，从而给人一种长条旗帜图片的错觉（不论视口有多宽都行）。这种方法为两处背景分别节省了 16KB（960 像素宽的`.png` 长条旗帜图一张 20KB，而精简的切片图只有 4KB）。这样你就为通过手机上网的用户节省了一部分流量！下图展示了这个切片图输出前的样子（放大到了 600%）：



在背景图片和文字大小都设置停当之后，And the winner isn't..网站在浏览器中的效果如下所示：



要让样式完美，还有很多工作要做。例如，导航按钮颜色应该交替使用红色和黑色，缺少内容区的 THESE SHOULD HAVE WON 按钮和侧边栏的 full info 按钮，字体与设计图效果相去甚远。不过，所有这些问题都可以使用 HTML5 和 CSS3 来解决。使用 HTML5 和 CSS3，而不再是简单插入图片（如我们以前做过的那样），会使网站与我们的响应式目标步调一致。时刻谨记，我们要保证代码和数据都尽可能精简，以便为带宽有限的用户提供愉悦的体验。

2.3.3　小视口下的内容剪切

现在，我们把美学问题放在一边，重点关注一下当视口缩减至小于 960 像素的情况，此时正在制作的首页中会有一部分内容被切掉。



这还只是将视口缩减到 673 像素宽，想象一下网站在 iPhone 3GS 等设备上将会有多难看？iPhone 3GS 的显示屏大小只有 320 × 480 像素。看看下面的效果图：



咦，等等，这效果看着挺好，或者说还行……刚才你还说会难看的！这是因为 iOS 上的 Safari 浏览器默认是在 980 像素宽的画布上渲染页面，然后将画布缩小到与视口大小匹配。虽然得放大页面才能看清楚，但页面内容没有被切掉。怎么阻止 Safari 或其他移动浏览器做这样的默认处理呢？

2.4　阻止移动浏览器自动调整页面大小

iOS 和 Android 浏览器都基于 WebKit（http://www.webkit.org/）核心。这两种浏览器以及很多其他浏览器（如 Opera Mobile），都支持用 viewport meta 元素覆盖默认的画布缩放设置。只需要在 HTML 的`<head>`标签中插入一个`<meta>`标签。`<meta>`标签中可以设置具体的宽度（如像素值）或者缩放比例如 2.0（设备实际尺寸的两倍）。下面是一个将页面放大到设备实际尺寸两倍显示的 `meta` 标签的示例：

```
<meta name="viewport"  content="initial-scale=2.0,width=device-width" />
```

将这个"标签插入到我们的 HTML 中，如下面的代码所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="initial-scale=2.0,width=device-width" />
<title>And the winner isn't…</title>
```

然后，在 Android 中加载该页面，显示效果如下所示：



如你所见，这并不是我们最终想要的效果，但这个夸张的效果说明了我们讨论的问题。

安装 iOS 模拟器和 Android 模拟器

虽然真机测试无可替代，但还是可以使用 Android 和 iOS 模拟器。Windows、Linux 和 Mac 版的 Android 模拟器都可以免费下载，Android 软件开发工具包（SDK）也可以免费安装。下载地址是 http://developer.android.com/sdk/。不过得使用命令行安装，需要你有一颗勇敢的心[1]。iOS 模拟器是 Xcode 开发包（在 Mac App Store 中免费下载）的一部分，只能在 Mac OS X 上使用。一旦安装了 Xcode，你就可以在这个路径下找到模拟器：~/Developer/Platforms/iPhoneSimulator.platform/Developer/Applications iOS Simulator.app

① 最新 Windows 版能够可视化安装。

我们来分析一下上面所示的<meta>标签，以理解它的工作原理。name="viewport"属性不言而喻。content="initial-scale=2.0 的意思是将页面放大两倍（同理，0.5 表示缩小一半，3.0 表示放大 3 倍），同时 width=device-width 告诉浏览器页面的宽度应该等于设备宽度。

<meta>标签还可以控制页面可缩放的范围。下面的代码允许用户将页面最多放大至设备宽度的 3 倍，最小压缩至设备宽度的一半。

```
<meta name="viewport" content="width=device-width, maximum-scale=3, minimum-scale=
    0.5" />
```

你还可以禁止用户缩放，不过缩放是一个重要的辅助功能，所以在实践中很少禁用。

```
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
```

user-scalable=no 即是禁止缩放。

现在，我们将缩放比例设置为 1.0，这表示浏览器将按照其视口的实际大小来渲染页面。将宽度设置为设备宽度，意味着支持该特性的浏览器都将会按照设备宽度的实际大小来渲染页面。下面是我们最终将要使用的<meta>标签：

```
<meta name="viewport" content="width=device-width,initial-scale=1.0" />
```

在竖直的 iPad 上浏览页面，可以看到还是有一部分内容被剪切掉了，但已经不再是缩小版的页面了！这就是本节想要达到的目的。相信我，这是个进步！



> 鉴于 viewport meta 标签的使用越来越普遍，W3C 正尝试通过 CSS 引入同样的功能。可以访问 http://dev.w3.org/csswg/css-device-adapt/，了解新的`@viewport`声明。基本思路是不用在`<head>`标签中添加`<meta>`标签了，而是可以在 CSS 中编写`@viewport { width: 320px; }`声明，同样可以将浏览器视口宽度设置为 320 像素。有些浏览器已经支持这种语法（如 Opera Mobile），不过要使用私有前缀，如`@-o-viewport { width: 320px; }`。

2.5　针对不同视口宽度修正设计

设置 viewport meta 标签后，任何浏览器都不再缩放页面了，现在我们可以针对不同视口来修正设计效果。首先在 CSS 中为平板设备（如 iPad）增加媒体查询，竖直 iPad 的视口宽度是 768 像素（横向视口宽度是 1024 像素，此时页面渲染效果很理想）。

```
@media screen and (max-width: 768px) {
  #wrapper {
    width: 768px;
  }
  #header,#footer,#navigation {
    width: 748px;
  }
}
```

媒体查询针对视口宽度不大于 768 像素的情况，重新调整了外壳、头部、页脚以及导航等页面元素的宽度。下图展示了此时页面在 iPad 上的效果：



上图的效果让我很受鼓舞。页面内容没再被剪切，而是与 iPad 屏幕（或者其他视口不超过 768 像素的设备）恰好匹配。但还得解决导航区链接超出背景图和主内容区浮动在侧边栏之下（因为内容区太宽，在有限的空间内放不下）这两个问题。我们再修改一下 CSS 中的媒体查询，代码片段如下所示：

```
@media screen and (max-width: 768px) {
  #wrapper {
    width: 768px;
  }
  #header,#footer,#navigation {
    width: 748px;
  }
  #content,#sidebar {
    padding-right: 10px;
    padding-left: 10px;
    width: 728px;
  }
}
```

现在，侧边栏和内容区填满了页面，且两边各留有适当的内边距。但这样的效果并不引人注目。我想把内容放在前面，把侧边栏放在后面（侧边栏的重要程度一般不高）。如果说我正在尝试制作一个真正的响应式设计，那在此处我又犯了一个低级错误。

2.6　响应式设计中内容始终优先

我们想让设计在多平台多视口的情况下保留尽可能多的内容（而不是使用 `display: none` 或类似方法来隐藏部分内容），但也要意识到内容模块显示顺序的重要性。目前，页面中侧边栏和主内容区标签的顺序决定了侧边栏会显示在主内容区前面。显然，窄视口设备的用户应该先看到主内容，而后再看到侧边栏。

我们还可以（或许应该）将内容区移到导航区域之上。这样那些使用小视口设备的用户就可以先看到主内容。这样无疑是坚决贯彻“内容优先”原则的合理做法。但是，多数情况下每个页面顶部还是应该有导航区，所以我乐得只将 HTML 代码中的侧边栏和内容区位置互换一下，让内容区出现在侧边栏之前。当前代码结构如下：

```
<div id="sidebar">
  <p>here is the sidebar</p>
</div>
<div id="content">
  <p>here is the content</p>
</div>
```

互换位置后的代码如下：

```
<div id="content">
  <p>here is the content</p>
</div>
```

```
<div id="sidebar">
  <p>here is the sidebar</p>
</div>
```

虽然我们互换了标签位置，但页面在大视口中的显示效果没有变化，因为侧边栏和内容区分别使用了 `float:left` 和 `float:right` 属性。但是在 iPad 上，则变成了首先显示内容区，下面才是侧边栏。

在调整好标签结构的顺序之后，我还着手为 768 像素宽的视口追加并替换了一些样式。现在最新的媒体查询代码如下所示：

```
@media screen and (max-width: 768px) {
  #wrapper,#header,#footer,#navigation {
    width: 768px;
    margin: 0px;
  }
  #logo {
    text-align:center;
  }
  #navigation {
    text-align: center;
    background-image: none;
    border-top-color: #bfbfbf;
    border-top-style: double;
    border-top-width: 4px;
    padding-top: 20px;
  }
  #navigation ul li a {
    background-color: #dedede;
    line-height: 60px;
    font-size: 40px;
  }
  #content, #sidebar {
    margin-top: 20px;
    padding-right: 10px;
    padding-left: 10px;
    width: 728px;
  }
  .oscarMain {
    margin-right: 30px;
    margin-top: 0px;
    width: 150px;
    height: 394px;
  }
  #sidebar {
    border-right: none;
    border-top: 2px solid #e8e8e8;
    padding-top: 20px;
    margin-bottom: 20px;
  }
  .sideBlock {
    width: 46%;
    float: left;
```

```
    }
    .overHyped {
      margin-top: 0px;
      margin-left: 50px;
    }
  }
```

此处添加的代码只会对视口等于或小于 768 像素的显示屏设备起作用。视口更宽的设备会忽略这些代码。另外，因为这些样式放在文件的末尾，所以会覆盖之前的重名样式。结果就是视口大的设备上没有什么变化，而视口宽度为 768 像素的设备上最终的效果如下所示：



不消说，我们没准备拿什么设计大奖。但仅通过几行 CSS 媒体查询代码，就为不同的视口做出一个完全不同的布局来，这还是挺牛的。这些代码都是什么意思呢？

首先，使用媒体查询将所有内容区宽度置为全屏：

```
#wrapper,#header,#footer,#navigation {
    width: 768px;
    margin: 0px;
}
```

然后是一些美化页面元素及布局的样式。例如，如下代码片段改变了导航区大小、布局以及背景，这样平板用户可以更方便地选择导航条目：


```
#navigation {
  text-align: center;
  background-image: none;
  border-top-color: #bfbfbf;
  border-top-style: double;
  border-top-width: 4px;
  padding-top: 20px;
}
#navigation ul li a {
  background-color: #dedede;
  line-height: 60px;
  font-size: 40px;
}
```

现在，同样的内容可以根据视口大小以不同的布局来显示了。媒体查询很赞，不是吗？太值得庆祝一下了。当你开香槟的时候，我用 iPhone 看了看页面的效果……如下图所示：



2.7　媒体查询只是必要条件之一

我们回顾一下的前面所讲的内容。很显然我们的工作还远未结束，网站在 iPhone 的 320 像素宽的小视口中显示得很糟糕。媒体查询尽其所能，根据设备特性应用了对应的样式。但问题是，现有的媒体查询只覆盖了小范围的视口。视口宽度小于 768 像素的设备都将看到内容被剪切，而视口介于 768 像素到 960 像素之间的设备，则会使用未受媒体查询样式影响的原有样式，结果我们已经知道了，一旦视口宽度小于 960 像素，页面就无法匹配（作者抱头长叹）。

我们需要流动布局

如果针对已知的特定访问设备，可以单独使用媒体查询来制作理想的设计效果，我们已经见过专门适配 iPad 有多简单。但是这种策略有严重的缺陷，换句话说，它不能适应未来的变化。目前的情形是，页面捕捉到媒体查询设置的断点，然后布局发生变化。但在捕捉到下一个视口断点之前，页面静止不变。我们需要比这更好的策略。针对各种视口的排列组合编写对应的 CSS 样式，无法兼容未来可能出现的设备；而一个完美的设计，往往能在一定程度上适应未来的发展。在这点上我们目前的解决方案尚不完备。目前的效果更像是一个自适应设计，而不是我们想要的真正的响应式设计。我们的设计应该在突变之前保持灵动。要做到这点，需要将呆板的固定布局修改成灵活的流动布局。

2.8　小结

本章我们学习了什么是 CSS3 媒体查询，如何在 CSS 文件中引入媒体查询，以及如何使用媒体查询来制作响应式网页。讨论了如何让移动浏览器像桌面版浏览器一样渲染我们的页面，另外还谈到了在编写页面标签时要遵守“内容优先”的原则。此外还学习了在设计中使用图片时如何保证尽可能精简。

然而，我们也认识到媒体查询只能为我们提供自适应设计效果，不能真正实现响应式设计。对于响应式设计来说，媒体查询是必需的，而能让我们的设计在媒体查询设置的断点之间灵动显示的流动布局技术同样必需。创建基本的流动布局，保证页面在媒体查询断点之间的显示效果平滑流畅，是我们下一章将要讲述的内容。

拥抱流式布局 3

20 世纪 90 年代末，我刚学习做网页的时候，页面布局结构都是基于表格的。通常，屏幕上显示的各部分元素都是百分比宽度。例如，左侧的导航栏可能占据 20%的屏幕宽度，主内容区占据 80%。那时的浏览器视口还没有像今天这样天差地别，所以表格式布局在有限范围的视口中表现良好且能自由伸缩。没人太在意某些句子在不同大小的屏幕上效果不一样。不过，随着 CSS 的兴起，网页设计也更加接近印刷设计，很多人（包括我）开始使用固定的基于像素的布局，而基于百分比的布局逐年减少。

所有伟大的设计和思想，都会卷土重来。迷你车、烫发（我喜欢）以及喇叭牛仔裤都在多年沉寂后卷土重来。现在，轮到百分比布局方法上演王者归来了。

本章内容

- 理解为什么响应式设计需要百分比布局
- 将元素的固定像素宽度转换为百分比宽度
- 将文字的固定像素大小转换为等量的相对尺寸
- 理解如何找到任意元素的上下文
- 学习如何使图片平滑缩放
- 学习如何为不同的大小的屏幕提供不同的图片
- 学习如何让媒体查询与弹性图片及流式布局协同工作
- 使用 CSS 网格系统从头创建一个响应式布局

3.1 固定布局经不起未来考验

我前面说过，在“表格布局”的时代落幕之后，很少有需要使用百分比布局。一般情况下，我接到的要求都是让 HTML 和 CSS 可以完美匹配 950-1000 像素显示屏。如果布局使用了百分比宽度（如 90%），耳边很快就能听到抱怨：“我的显示器上效果不太一样。”固定像素尺寸的网页是匹配固定像素尺寸显示器的最简单办法。

即使在最近，当我们针对某个特定的流行设备如 iPad 或者 iPhone，使用媒体查询来制作

修正版的布局时（类似第 2 章的例子），仍然是基于已知的视口宽度使用固定像素尺寸。很多人喜欢这种方式，因为每次有新东西出来，只要用户要求修改网站，就可以借机收一次费。但是，这种方法不是一种完全兼容未来的网页制作方法。未来，还会出现更多大小不一的视口，我们需要一些适应未知设备的方法。

3.2 为什么响应式设计需要百分比布局

在认识到媒体查询威力无比的同时，我们也要看到它的局限性。那些仅使用媒体查询来适应不同视口的固定宽度设计，只会从一组 CSS 媒体查询规则突变到另一组，两者之间没有任何平滑渐变。从我们在第 2 章的例子来看，当某个视口处于媒体查询设置的固定宽度范围之外（可能是某种未知的未来设备及视口），网页就需要水平滚动才能完整浏览。不过我们想要的是一个灵活的设计，能在所有视口中都完美显示，而不仅仅只针对媒体查询设定的一些固定视口。切入主题吧（看出来我在这抖的包袱了吗？我用了一句电影行话来搭配我们的电影主题的网站……没看懂？好冷啊，我去穿件外套……），我们需要将固定像素布局转换成灵活的百分比布局。这样才能让页面元素根据视口大小在一个又一个媒体查询之间灵活伸缩修正样式。

> **百分比布局和媒体查询和谐共处**
>
> 我之前提到过伊桑·马科特在 A List Apart 上发表的有关响应式网页设计的文章（http://www.alistapart.com/articles/responsive-web-design/）。其实他采用的技术（流动布局、弹性图片和媒体查询）并不新颖，而将这些理念基于一套完整统一的方法论进行应用则很有创造性。对于很多网页设计工作来说，他的文章打开了很多新的思路。事实上，这种网页设计的新方法做到了两全其美：使用百分比布局创建流动的弹性界面，同时使用媒体查询来限制元素的变动范围。将这两者组合到一起构成了响应式设计的核心，基于此可以创造出真正完美的设计。

3.3 将网页从固定布局修改为百分比布局

在可预见的未来，你看到的和制作的网页都还会使用固定尺寸。当前，我们通常是在 Photoshop、Fireworks 等软件制作的设计图中（基于像素单位）度量元素的大小、外边距，然后将这些尺寸直接写进 CSS 代码中。文字大小也是这样设置的。我们在图片编辑软件中点击一下文字对象，查看其具体尺寸（一般单位是像素），然后将其写入对应的 CSS 代码。那我们如何将固定尺寸转换为相对尺寸呢？

3.3.1　需要牢记的公式

作为伊桑·马科特的粉丝，我对他的崇拜可能有点太过了。但此处我有必要再给他行一次三跪九叩之礼。在丹·锡德霍姆（Dan Cederholm）编写的《无懈可击的 Web 设计》一书中，伊桑·马科特为其撰写了一章关于流动布局的内容。在书中，他提供了一个简易可行的公式，将固定像素宽度转换对应的百分比宽度：

目标元素宽度÷上下文元素宽度=百分比宽度

看起来有点像方程式？不要害怕，在创建响应式设计的过程中，这个公式很快会成为你的得力助手。不再赘述理论，接下来我们使用该公式将 And the winner isn't..网站从固定尺寸转换成百分比布局。

如果你还记得，我们在第 2 章学习使用媒体查询来支持不同视口时，所编写的页面基本标签结构应该是这样的：

```
<div id="wrapper">
  <!-- the header and navigation -->
  <div id="header">
    <div id="navigation">
      <ul>
        <li><a href="#">navigation1</a></li>
        <li><a href="#">navigation2</a></li>
      </ul>
    </div>
  </div>
   <!-- the sidebar -->
  <div id="sidebar">
    <p>here is the sidebar</p>
  </div>
  <!-- the content -->
  <div id="content">
    <p>here is the content</p>
  </div>
  <!-- the footer -->
  <div id="footer">
    <p>Here is the footer</p>
  </div>
</div>
```

页面内容稍后添加，但我们要重点关注的是此处用来设置页面主要结构模块（头部、导航、侧边栏、内容区以及页脚）宽度的 CSS。请注意，下面的代码中我省去了很多样式，以便将注意力集中在页面结构上：

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 960px;
}
```

```
#header {
  margin-right: 10px;
  margin-left: 10px;
  width: 940px;
}

#navigation {
  padding-bottom: 25px;
  margin-top: 26px;
  margin-left: -10px;
  padding-right: 10px;
  padding-left: 10px;
  width: 940px;
}

#navigation ul li {
  display: inline-block;
}

#content {
  margin-top: 58px;
  margin-right: 10px;
  float: right;
  width: 698px;
}

#sidebar {
  border-right-color: #e8e8e8;
  border-right-style: solid;
  border-right-width: 2px;
  margin-top: 58px;
  padding-right: 10px;
  margin-right: 10px;
  margin-left: 10px;
  float: left;
  width: 220px;
}

#footer {
  float: left;
  margin-top: 20px;
  margin-right: 10px;
  margin-left: 10px;
  clear: both;
  width: 940px;
}
```

上面代码中所有的尺寸值都是像素值。我们从最外层的元素开始，使用“目标元素宽度÷上下文元素宽度=百分比宽度”这个公式将它们改成百分比宽度。

当前所有的页面内容都被包裹在一个 ID 为`#wrapper` 的 div 中。在上面所示的 CSS 中可见，这个 div 被设置为外边距自适应，宽度 960 像素。作为最外层的 div，我们该把它的宽度定义为相对视口宽度的百分之多少呢？

3.3.2 设置百分比元素的上下文

我们需要一些“hold”场面的元素，让它们作为网页中百分比宽度元素（内容区、侧边栏、页脚）的上下文。我们要为这些元素的宽度设置一个百分比值，而#wrapper 元素的宽度应该是相对于视口尺寸而言的。现在我们从头来过，将其宽度设置为 96%看看效果如何。#wrapper 元素修改后的样式如下：

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 96%; /* 控制最外层的div */
}
```


修改后的效果如下图所示：



看起来还不错！96%的宽度刚刚好，不过也可以选择 100%或者 90%——这取决于我们的感觉，保证网页在视口内有最美观的视觉效果即可。

将里层元素从固定宽度改为百分比宽度稍微有点复杂。我们先来看看头部。再复习一下公式：目标元素宽度÷上下文元素宽度=百分比宽度。#header（目标元素）被包裹在#wrapper（上下文元素）中。因此，我们用#header 的宽度 940 像素，来除以上下文元

素（`#wrapper`）的宽度 960 像素，结果是 0.979166667。将小数点向右移动两位转换为百分比值，我们就得到了头部的百分比宽度，即 97.9166667。将结果设置到 CSS 中：

```
#header {
  margin-right: 10px;
  margin-left: 10px;
  width: 97.9166667%; /* 940 ÷ 960 */
}
```

`#navigation` 和`#footer` 的宽度也是基于像素的，我们使用相同的方法将它们转换为百分比。

最后在浏览器中查看效果之前，我们来说说`#content` 和`#sidebar`。上下文宽度还是 960 像素，那我们只需要用目标元素宽度来除这个数字即可。`#content` 的宽度是 698 像素，除以 960 结果是 0.727083333。移动小数点后结果是 72.7083333%——这就是`#content` 的百分比宽度。侧边栏的宽度是 220 像素，但它有 2 像素宽的边框。我不想让右侧边框的宽度随着上下文变宽或是变窄，而是始终保持在 2 像素。所以需要将侧边栏的宽度减去一点。我将本例中的侧边栏宽度减去 2 像素，然后套用之前的公式，即目标元素宽度（218 像素）除以上下文元素宽度（960 像素），结果是 0.227083333。移动小数点后，侧边栏的百分比宽度是 22.7083333%。再将所有像素宽度修改为百分比宽度，最终的 CSS 代码如下所示：

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 96%; /* 最外层 DIV */
}

#header {
  margin-right: 10px;
  margin-left: 10px;
  width: 97.9166667%; /* 940 ÷ 960 */
}

#navigation {
  padding-bottom: 25px;
  margin-top: 26px;
  margin-left: -10px;
  padding-right: 10px;
  padding-left: 10px;
  width: 72.7083333%; /* 698 ÷ 960 */
}

#navigation ul li {
  display: inline-block;
}

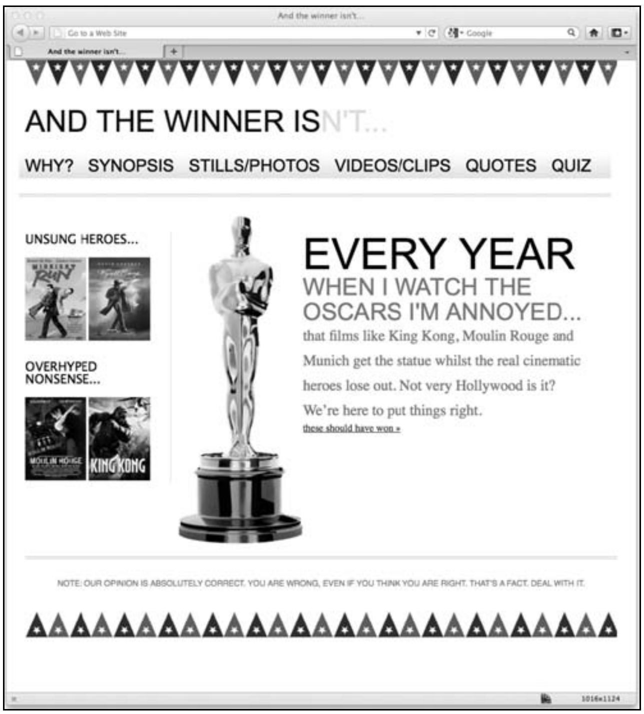
#content {
  margin-top: 58px;
```

```
  margin-right: 10px;
  float: right;
  width: 72.7083333%; /* 698 ÷ 960 */
}

#sidebar {
  border-right-color: #e8e8e8;
  border-right-style: solid;
  border-right-width: 2px;
  margin-top: 58px;
  margin-right: 10px;
  margin-left: 10px;
  float: left;
  width: 22.7083333%; /* 218 ÷ 960 */
}

#footer {
  float: left;
  margin-top: 20px;
  margin-right: 10px;
  margin-left: 10px;
  clear: both;
  width: 97.9166667%; /* 940 ÷ 960 */
}
```


下图展示了当前页面在视口宽度约为 1000 像素的 Firefox 浏览器中的效果：



目前看来效果很不错。接下来我们继续使用“目标元素宽度÷上下文元素宽度=百分比宽度”这个公式，将页面各处 10 像素的内边距、外边距也替换成等价的百分比值。所有这些间距都基于 960 像素的上下文，所以替换成对应的百分比值就是 1.0416667%（10 ÷ 960）。

> **这些数字能四舍五入吗？**
>
> 一些响应式设计技术的批评者（如这篇文章：http://tripleodeon.com/2010/10/not-a-mobile-web-merely-a-320px-wide-one/）认为在样式表中输入诸如.550724638 这样的数字很愚蠢。你可能也会疑惑，为什么不将这些小数四舍五入？但支持者们认为，这样做可以提供更加精确的结果。为浏览器提供更加精确的结果可以使其显示效果更加精准。顺便说一下，学过数学的人都应该知道黄金分割率（http://en.wikipedia.org/wiki/Golden_ratio）。黄金分割率是一个很早就被发现且广泛应用于各学科的数学比例，其比值大约为 1:1.61803398874989（如果你想知道保留 10000 位小数的黄金分割数，请访问这里：http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/ Fibonacci/phi10000dps.txt）。怎么看它都不是一个简洁的数字，但是它非常重要。黄金分割率的测量都能做到如此精确，那我相信网页设计同样做得到。

在如前所述的视口中，页面元素表现良好。但是导航区域不太理想。如果将视口尺寸缩小一点，只要一丁点，导航链接就会变成两行：



此外，如果将视口放大，导航链接之间的间距并不会相应地增加。我们来看看与导航关联的 CSS 代码并试着找出问题的原因：

```
#navigation {
  padding-bottom: 25px;
  margin-top: 26px;
  margin-left: -1.0416667%; /* 10 ÷ 960 */
  padding-right: 1.0416667%; /* 10 ÷ 960 */
  padding-left: 1.0416667%; /* 10 ÷ 960 */
  width: 97.9166667%; /* 940 ÷ 960 */
  background-repeat: repeat-x;
  background-image: url(../img/atwiNavBg.png);
  border-bottom-color: #bfbfbf;
  border-bottom-style: double; border-bottom-width: 4px
}

#navigation ul li {
   display: inline-block;
}

#navigation ul li a {
  height: 42px;
  line-height: 42px;
  margin-right: 25px;
  text-decoration: none;
  text-transform: uppercase;
  font-family: Arial, "Lucida Grande", Verdana, sans-se
  font-size: 27px;
  color: black;
}
```


一眼看上去，就发现好像上面的第三个样式规则`#navigation ul li a`，还有 25px 这样的基于像素单位的外边距。让我们用久经考验的公式来修正这个问题。导航区域的宽度是 940 像素，所以换算结果是 2.6595745%。对该样式的修改如下：

```
#navigation ul li a {
  height: 42px;
  line-height: 42px;
  margin-right: 2.6595745%; /* 25 ÷ 940 */
  text-decoration: none;
  text-transform: uppercase;
  font-family: Arial, "Lucida Grande", Verdana, sans-serif;
  font-size: 27px;
  color: black;
}
```

够简单吧！我们到浏览器中看看是否万事如意……



哎，等等，这不是我们想要的效果。导航链接没再折成两行，但很明显导航链接之间的间距不对。导航链接看上去像一个字典里查不到的长单词……

3.3.3　必须时刻牢记上下文

再想想我们的公式（目标元素宽度÷上下文元素宽度=百分比宽度），你就会知道前述问题的症结所在——上下文元素。相关的标签结构如下：

```
<div id="navigation">
  <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
  </ul>
</div>
```

可以看到<a href="#">链接被包裹在各自对应的<li>标签中。它们才是我们要找的外边距的上下文元素。看看<li>标签对应的 CSS 代码，会发现没有为其设置宽度：

```
#navigation ul li { display: inline-block; }
```

这种情况很常见，不过我们有很多方法解决这个问题。我们可以给<li>标签设置一个明确的宽度值，这个值必须得是一个固定像素宽度或者是一个相对于其包裹元素（#navigation div）的百分比值，但这两种值都无法保证<li>中的文字灵活可变。

此外我们可以修改<li>现在的 CSS 代码，将 inline-block 改为 inline：

```
#navigation ul li {
  display: inline;
}
```

3

使用 display: inline;（这样可以阻止<li>元素渲染为块级元素）还可以使导航在不支持 inline-block 的老版本 Internet Explorer（版本 6 和 7）中水平显示。不过，我是 inline-block 的粉丝，因为它允许我们在现代浏览器中有效的控制外边距和内边距，所以我会让<li>标签继续保持 inline-blocks 状态（一会可以给 IE6 和 IE7 追加一个覆写样式），然后将（没有明确的上下文的）<a>标签上的百分比外间距挪到<li>上来。修改后的样式如下所示：

```
#navigation ul li {
  display: inline-block;
  margin-right: 2.6595745%; /* 25 ÷ 940 */
}

#navigation ul li a {
  height: 42px;
  line-height: 42px;
  text-decoration: none;
  text-transform: uppercase;
  font-family: Arial, "Lucida Grande", Verdana, sans-serif;
  font-size: 27px;
  color: black;
}
```

下图展示了在视口宽度 1200 像素的浏览中的显示效果。

导航区域的修改现在告一段落，但是导航链接在视口变小时仍然会折成两行，只有到了视口小于 768 像素时，第 2 章编写的媒体查询才能覆写当前的导航样式。在着手修正导航部分的问题之前，我准备先将文字大小从像素尺寸修改为相对单位 em。完成这个工作之后，我们就会注意到另外一个被忽略的问题，即让图片随着页面的变化缩放。



3.4　用 em 替换 px

过去的几年里，网页设计师使用 em 替代 px 主要是为了文字缩放。因为老版本的 Internet Explorer 无法缩放像素单位的文字。不过现代浏览器很久以前就支持缩放以像素为单位的文字了。那用 em 替换 px 还有什么必要性或优越性呢？有两点显而易见的理由：一是那些使用 Internet Explorer 6 的用户也将能够缩放文字，二是这样做可以使我们设计师和开发者的生活更简单。em 的实际大小是相对于其上下文的字体大小而言的。如果我们给 <body>标签设置文字大小为 100%，给其他文字都使用相对单位 em，那这些文字都会受 body 上的初始声明的影响。这样做的好处就是，如果在完成了所有文字排版后，客户又提出将页面文字统一放大一点，我们就可以只修改 body 的文字大小，其他所有文字也会相应变大。

同样，“**目标元素尺寸÷上下文元素尺寸=百分比尺寸**”这个公式也适用于将文字的像素单位转换为相对单位。值得注意的是，现代浏览器的默认文字大小都是 16 像素（显式声明的除外）。因此一开始给 body 标签应用下列任何一条规则所产生的效果都一样：

```
font-size: 100%;
font-size: 16px;
font-size: 1em;
```

下面举一个例子。示例网站的样式表中，第一段以像素为单位的文字就是页面左上角的网站标题：AND THE WINNER ISN’T…

```
#logo {
  display: block;
  padding-top: 75px;
  color: #0d0c0c;
  text-transform: uppercase;
  font-family: Arial, "Lucida Grande", Verdana, sans-serif
  font-size: 48px;
}

#logo span { color: #dfdada; }
```

因为 48 ÷ 16 = 3，所以我们将样式修改如下：

```
#logo {
  display: block;
  padding-top: 75px;
  color: #0d0c0c;
  text-transform: uppercase;
  font-family: Arial, "Lucida Grande", Verdana, sans-serif;
  font-size: 3em; /* 48 ÷ 16 = 3*/
}
```

这个逻辑在整个网站中通用。如果发现哪出了毛病，那应该是目标元素的上下文发生了变化。以页面中的标签为例：

```
<h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
```

修改后的相对单位样式如下：

```
#content h1 {
  font-family: Arial, Helvetica, Verdana, sans-serif;
  text-transform: uppercase;
  font-size: 4.3125em; } /* 69 ÷ 16 */

#content h1 span {
  display: block;
  line-height: 1.052631579em; /* 40 ÷ 38 */
  color: #757474;
  font-size: .550724638em; /* 38 ÷ 69 */
}
```

可以看到`<span>`元素的文字大小（38 像素）是相对于其父元素的文字大小（69 像素）而言的。而它的行高（40 像素）则是相对于其本身的文字大小（38 像素）而言。

现在，我们的页面结构可以自动缩放，文字大小也已从像素单位转换成相对单位。但是，我们还必须解决图片大小随视口缩放的问题。那我们接下来看看这个问题。

> **em 究竟是什么？**
>
> em 是书面形式的大写字母“M”的简称，发音和 M 相同。以前，“M”常被用来测定某种字体的大小，因为它是英文字母中最大（最宽）的字母。如今，em 作为一个测量单位，指的是特定字母的宽度和高度相对于特定字体磅值的比例。

3.5　弹性图片

在现代浏览器（包括 IE 7+）中要实现图片随着流动布局相应缩放非常简单。只需在 CSS 中作如下声明：

```
img {
  max-width: 100%;
}
```

这样就可以使图片自动缩放到与其容器 100%匹配。更进一步，可以将同样的样式应用到其他多媒体标签上。如：

```
img,object,video,embed {
  max-width: 100%;
}
```

这些多媒体元素都可以自动缩放了。但是，对于采用的`<iframe>`显示视频的网站（比如 YouTube），这个技术还不行，我们会在第 4 章解决这个问题。眼下，还是专注于图片的缩放问题吧，因为不论何种多媒体元素，原理都是相通的。

使用这种方法有几个需要斟酌的问题。第一，要提前准备一张足够大的图片，以备大视口使用。但这也就引出了第二个，同时也是非常重要的问题，即无论视口多大，什么设备，都得下载超大图片。可能对某些设备来说，图片大小只要原始图片的 25%就好了。另外，在某些情况下，你还不得不因此而考虑带宽限制。我们呆会儿再说第二个问题，先把图片缩放问题说完。

3.5.1　让图片随视口缩放

示例网站的侧边栏有一组电影海报，其中两部是好电影，两部是烂电影（不在此处讨论这个话题）。对应的标签结构如下：

```
<!-- the sidebar -->
  <div id="sidebar">
    <div class="sideBlock unSung">
      <h4>Unsung heroes...</h4>
      <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" width="99"height=
         "135" /></a>
      <a href="#"><img src="img/wyattEarp.jpg" alt="Wyatt Earp" width="99" height=
```

```
            "135" /></a>
      </div>
      <div class="sideBlock overHyped">
        <h4>Overhyped nonsense...</h4>
        <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" width="99" height=
            "135" /></a>
        <a href="#"><img src="img/kingKong.jpg" alt="King Kong" width="99" height=
            "135" /></a>
      </div>
    </div>
```

尽管我在 CSS 文件中给 `img` 元素追加了 `max-width: 100%`声明，但是没看到什么变化，图片没有按我预想的方式随着视口变化而变化：



原因是我在标签中指定了图片的宽度和高度：

```
<img src="img/wyattEarp.jpg" alt="Wyatt Earp" width="99" height="135" />
```

又犯了一个低级错误！修改对应的图片标签，删除宽度和高度属性：

```
<img src="img/wyattEarp.jpg" alt="Wyatt Earp" />
```

刷新浏览器看看修改的效果：



修改确实起效了！但却引入了另外一个问题。因为图片被缩放至其包裹元素宽度的 100%，所以每张图片都填满了侧边栏。同样，解决这个问题的方法有很多……

3.5.2　为特定图片指定特定规则

可以像下面代码所示的那样给每个图片追加一个额外的 class：

```
<img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt Earp" />
```

然后，为其宽度设置一个特定的规则。另外，我们还可以保留图片标签不变，利用 CSS 规则的针对性，用一个更具体的规则覆写侧边栏图片已有的 max-width 样式：

```
img {
  max-width: 100%;
}

.sideBlock img {
  max-width: 45%;
}
```

下图展示了侧边栏当前的显示效果：



这种利用 CSS 规则针对性的方法，也可以用来控制其他图片或多媒体元素的宽度。第 5 章会介绍 CSS3 选择器的强大威力，它能让我们不追加任何多余标签，或是不引入任何 JavaScript 框架（如 jQuery），直接指向页面任意元素，来完成我们的苦活累活。

我给侧边栏图片设置的宽度是 45%，因为图片之间需要一点间距，所以两张图片宽度加起来是 90%，给我留出了一点余地（10%）。

现在侧边栏图片终于完美显示了，我准备将奥斯卡雕像图片的标签中的宽度和高度也删除掉。但是，除非我为其设置一个百分比宽度，否则它还是不会自动缩放；所以我再次使用我们值得信赖的公式来为其设置一个百分比宽度。

```
.oscarMain {
  float: left;
  margin-top: -28px;
  width: 28.9398281%; /* 202 ÷ 698 */
}
```

3.5.3 给弹性图片设置阈值

现在图片可以随着视口的伸缩而缩放了。但是如果将视口拉大，直到图片拉伸至超出其原始尺寸，那问题就麻烦了。看看下图所示的 1900 像素宽的视口中奥斯卡图片的显示效果：



oscar.png 图片的实际宽度是 202 像素。但在超过 1900 像素宽的视口中，图片也被拉大了，其显示宽度超过了 300 像素。我们可以通过追加另一个特定样式来为图片设置阈值：

```
.oscarMain {
  float: left;
  margin-top: -28px;
  width: 28.9398281%; /* 698 ÷ 202 */
  max-width: 202px;
}
```

这样就可以保证 oscar.png 按照通用的图片样式自由缩放，但又绝不会超出 `max-width` 属性设置的最大上限。下图展示了给图片设置上限之后的页面效果：



3.5.4　超级全能的 max-width 属性

另一种限制页面无限制扩张的方法是给最外层的#wrapper div 设置 max-width 属性，如下所示：

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 96%; /* Holding outermost DIV */
  max-width: 1414px;
}
```


这样意味着页面会缩放至视口宽度的 96%，但绝不会超过 1414 像素宽（设置为 1414 像素，是因为在这个宽度下大多数浏览器恰好能剪切出完整的角旗，而不会留个半边）。下图展示了视口宽度 1900 像素下的页面效果：



显然这些方法仅是备用选项。但它证明了流动布局的灵活性，也说明了如何通过几个特定声明来控制文档流。

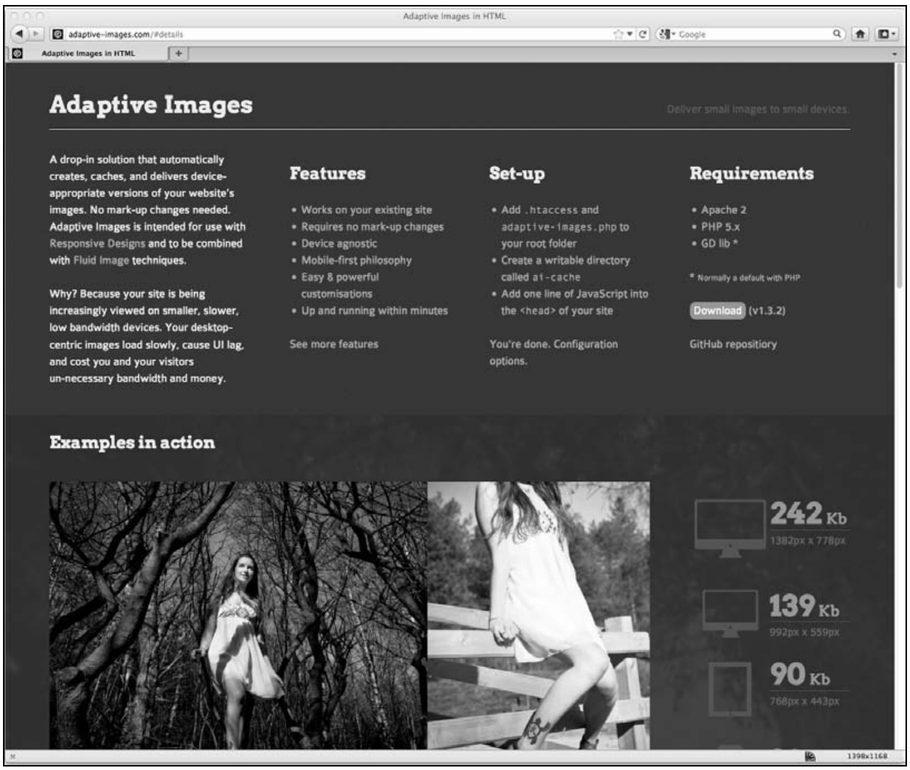
3.6　为不同的屏幕尺寸提供不同的图片

现在我们可以让图片完美缩放，而且也知道了如何限制特定图片的显示尺寸。但在这章的前面我们曾指出图片缩放存在一定的问题。图片尺寸必须比其显示尺寸更大以保证渲染效果，否则的话图片可能看起来很糟糕。基于这个原因，图片文件的体积就永远比实际显示所需的大。

很多人都在研究这个问题，尝试为较小的屏幕尺寸提供较小的图片。第一个著名的例子是 Filament Group 的“响应式图片”(http://filamentgroup.com/lab/responsive_images_experimenting_with_context_aware_image_sizing/)。不过最近，我已经转而使用 Matt Wilcox 的“自适应图片”(http://adaptive-images.com)了。Filament Group 的解决方案需要对图片标签做一定修改。Matt Wilcox 的解决方案则不需要，而且他的方案会根据标签中已经设定的全尺寸图片自动创建各种尺寸的图片。这种解决方案允许基于一组屏幕尺寸断点，根据用户需要为其提供不同的图片。接下来我们就看一看如何利用该技术实现图片自适应。

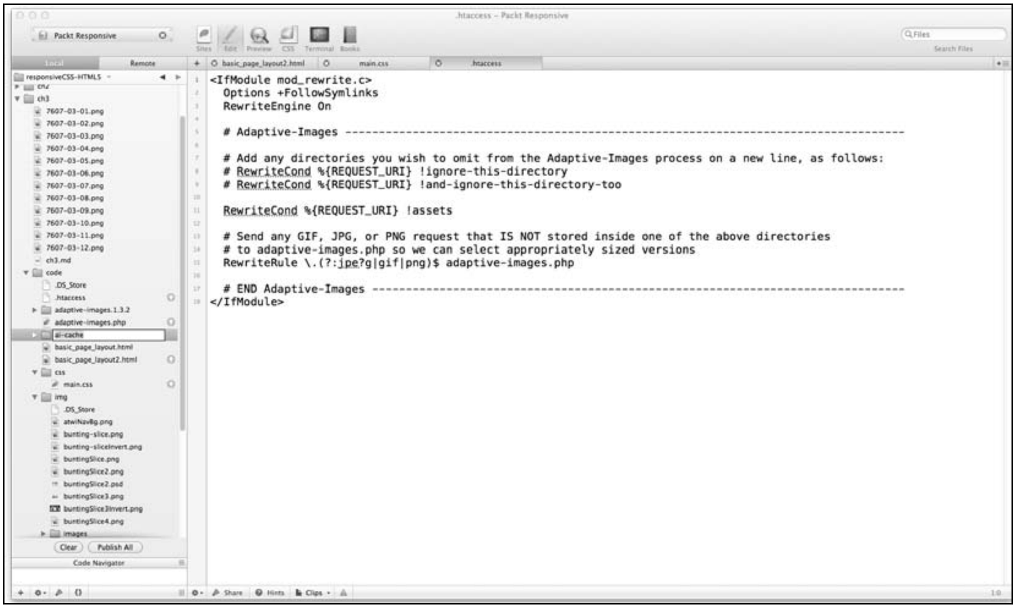
设置自适应图片

实现 Adaptive Images 解决方案需要 Apache 2、PHP 5.x 和 GD 库，也就是说需要 Web 服务器端编程。首先，在其网站上下载.zip 文件开始配置：

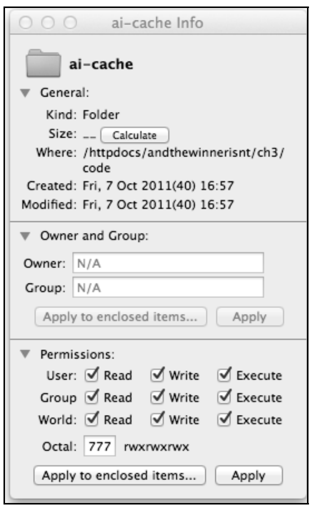


解压文件，然后将其中的 `adaptive-images.php` 和 `.htaccess` 文件拷贝到网站的根目录。如果你网站的根目录下已经有一个 `.htaccess` 文件了，不要覆盖它。参考下载包中的 `instructions.htm` 文件看看怎么做合适。

接着在网站根目录下创建一个名为 ai-cache 的文件夹。



用你最喜欢的 FTP 客户端软件设置该文件夹的权限为 777。



然后把如下 JavaScript 代码复制到每个需要自适应图片的网页的头部：

```
<script>document.cookie='resolution='+Math.max(screen.width,screen.height)+';
path=/';/script>
```

如果你没有使用 HTML5（在下一章会改用 HTML5），想让页面通过标准验证，则需要追加 type 属性。所以 script 标签应如下所示：

```
<script type="text/javascript">document.cookie='resolution='+Math.maxscreen.width,
screen.height)+'; path=/';</script>
```

切记这段 JavaScript 代码要放在页面头部（最好作为第一个脚本），因为它需要在页面加载完成之前，而且要在发出图片请求之前运行。下面是我们的示例网站头部加入该脚本后的结果：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport"  content="width=device-width,initial-scale=1.0" />
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.
max(screen.width,screen.height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
```

把背景图片放在其他地方

过去，我通常将所有图片都放在一个名如 images 或 img 的文件夹中，不论是用做 CSS 背景的图片，还是通过标签插入的图片。但是在使用自适应图片方案时，建议将那些用于 CSS 的背景图片（或者那些你不想被缩放的图片）放在另一个目录。自适应图片方案默认为此创建的目录是 `assets`。如果你不想缩放某张图片，把它丢进这个文件夹即可。如果你想将这类图片存在其他（或更多）文件夹中，则需要像下面这样修改 `.htaccess` 文件。

```
<IfModule mod_rewrite.c>
  Options +FollowSymlinks
  RewriteEngine On
  # Adaptive-Images --------------------------------------------------------

  RewriteCond %{REQUEST_URI} !assets
  RewriteCond %{REQUEST_URI} !bkg

  # Send any GIF, JPG, or PNG request that IS NOT stored inside one of the above
directories
  # to adaptive-images.php so we can select appropriately sized versions
  RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php
  # END Adaptive-Images ----------------------------------------------------
</IfModule>
```

上面的代码设定了存在 `assets` 或 `bkg` 文件夹中的图片不会被缩放。反之，如果你想显式声明只允许某个特定文件夹中的图片被缩放，那么将设置规则中的感叹号去掉即可。

例如，如果我只想让网站根目录下名为 andthewinnerisnt 的文件夹中的图片被缩放，则修改后的代码如下所示：

```
<IfModule mod_rewrite.c>
  Options +FollowSymlinks
  RewriteEngine On

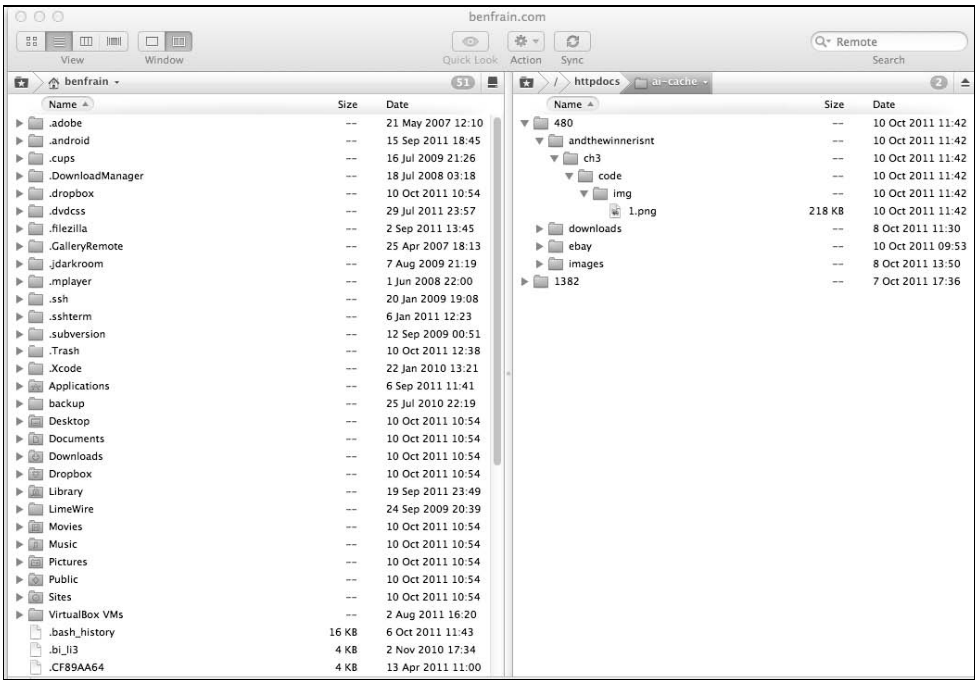
  # Adaptive-Images ------------------------------------------------------

  RewriteCond %{REQUEST_URI} andthewinnerisnt

  # Send any GIF, JPG, or PNG request that IS NOT stored inside one of the above directories
  # to adaptive-images.php so we can select appropriately sized versions
  RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php

  # END Adaptive-Images --------------------------------------------------
</IfModule>
```


这就是要设置的全部内容。检查是否能正确生成自适应图片的最简单办法，就是在网页中插入一张大图片，然后用手机访问这个页面。之后用 FTP 软件检查 ai-cache 文件夹中的内容，你应该可以看到一堆文件，以及使用屏幕尺寸断点命名的文件夹，如 480（如下图）：



Adaptive Images 方案不仅限于静态网站，它也可以被用于内容管理系统，而且在 JavaScript

被禁用的情况下依然有效。自适应图片方案给我们提供了一种方法，可以根据屏幕尺寸提供完全不同的图片，为那些没有必要下载全尺寸大图的设备节省带宽。

3.7 流动网格布局和媒体查询的默契配合

在本章的前面，我们的导航链接在特定的视口宽度下回折成两行。我们可以使用媒体查询来修正这个问题。如果导航链接在 1060 像素下散得太开而在 768 像素下显示正常（我们之前的媒体查询做好了处理），我们就给这个视口范围设置一下字体样式：

```
@media screen and (min-width: 1001px) and (max-width: 1080px) {
    #navigation ul li a { font-size: 1.4em; }
}
@media screen and (min-width: 805px) and (max-width: 1000px) {
    #navigation ul li a { font-size: 1.25em; }
}
@media screen and (min-width: 769px) and (max-width: 804px) {
    #navigation ul li a { font-size: 1.1em; }
}
```

你看到了，我们根据视口宽度来改变文字大小，结果就是这组导航链接在 769 像素到无穷大的视口中都会显示在一行。这是媒体查询和流动布局和谐共存的又一证据：媒体查询约束流动布局的变动范围，而流动布局则简化了从一组媒体查询样式过渡到另一组的改变过程。

3.8 CSS 网格系统

人们对 CSS 网格系统/框架的看法褒贬不一，有人夸赞它，有人咒骂它。为了不收到满怀怨恨的邮件，我声明我保持中立。我能理解一些开发者认为网络系统是多余的，某些情况下还会产生一些冗余代码，但同时我也很欣赏它们在快速搭建界面布局上的价值。

下面是一些对响应式设计提供了不同程度支持的 CSS 框架：

- Semantic (http://semantic.gs)；
- Skeleton (http://getskeleton.com)；
- Less Framework (http://lessframework.com)；
- 1140 CSS Grid (http://cssgrid.net)；
- Columnal (http://www.columnal.com)。

这些框架中，我个人最喜欢 Columnal 网格系统，因为它有一套内置的集成了媒体查询的流动网格布局，而且它使用了与 960.gs 框架类似的 CSS 命名，960.gs 框架是一套广为开发者和设计师所熟知的非常流行的固定宽度网格系统。

> **Alpha, Omega 以及其他常用的网格类名**
>
> 很多 CSS 网格系统都使用一些特定的 CSS 类来完成日常的布局工作。`row` 和 `container` 类不言自明，但经常会有很多类需要说明。因此，时常查阅网格系统说明文档中有关这些类的说明，能让我们的工作更加轻松。例如，网格系统中常用的典型 CSS 类有 `alpha` 和 `omega`——分别表示一行中的第一个和最后一个元素（`alpha` 和 `omega` 类中会去除内边距或外边距），还有`.col_x`，其中 x 指的是该元素横跨合并的列数（如 `col_6` 表示横跨 6 列）。

使用网格系统快速搭建网站

假设我们还没有构建流动布局，也还没有编写任何媒体查询代码。我们手里只有 And the winner isn't...网站首页的 PSD 分层图，且被要求使用 HTML 和 CSS 尽可能快地搭建起网站的基本布局结构。我们来看看 Columnal 网格系统如何帮助我们快速实现这个目标。

在 PSD 原始设计图中，很容易看出网站是 16 列布局。但 Columnal 网格系统最大只支持 12 列，所以我们将 PSD 中的 16 列改为 12 列。



下载 Columnal 网格系统的压缩包然后解压，将我们已有的页面复制一份，然后将页面头

部的 CSS 文件链接从 `main.css` 改为 `columnal.css`。使用 Columnal 系统创建页面结构的关键是给 div 添加正确的 class。下面的代码是目前为止页面的完整标签结构：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport"  content="width=device-width,initial-scale=1.0"/>
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,
    screen.height)+'; path=/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />

</head>

<body>

<div id="wrapper">
  <!-- the header and navigation -->
  <div id="header">
    <div id="logo">And the winner is<span>n't...</span></div>
    <div id="navigation">
      <ul>
        <li><a href="#">Why?</a></li>
        <li><a href="#">Synopsis</a></li>
        <li><a href="#">Stills/Photos</a></li>
        <li><a href="#">Videos/clips</a></li>
        <li><a href="#">Quotes</a></li>
        <li><a href="#">Quiz</a></li>
      </ul>
    </div>
  </div>
  <!-- the content -->
  <div id="content">
    <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
    <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
    <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst
        the real cinematic heroes lose out. Not very Hollywood is it?</p>
    <p>We're here to put things right. </p>
  <a href="#">these should have won &raquo;</a>
  </div>
  <!-- the sidebar -->
  <div id="sidebar">
    <div class="sideBlock unSung">
      <h4>Unsung heroes...</h4>
      <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" /></a>
      <a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt
          Earp" /></a>
    </div>
    <div class="sideBlock overHyped">
      <h4>Overhyped nonsense...</h4>
      <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" /></a>
      <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
```

```
    </div>
  </div>
  <!-- the footer -->
  <div id="footer">
    <p>Note: our opinion is absolutely correct. You are wrong, even if you think
        you are right. That's a fact. Deal with it.</p>
  </div>

</div>
</body>
</html>
```

首先，要将#wrapper div 设置为其他所有元素的容器，所以为其追加.container 类：

```
<div id="wrapper" class="container">
```

接着往下看，AND THE WINNER ISN'T 是页面的第一行文字，所以为头部追加.row 类：

```
<div id="header" class="row">
```

网页 logo 虽然只是文字，嵌在一行里，但横跨 12 列。因此为其追加.col_12 类：

```
<div id="logo" class="col_12">And the winner is<span>n't...</span></div>
```

紧接着一行是导航，所以为其追加一个.row 类：

```
<div id="navigation" class="row">
```

然后继续根据需要给元素追加.row 和.col_x 类。此处我将跳过后续的修改，因为我觉得重复这个过程可能会让你打盹。下面展示的代码就是修改后的完整页面标签。注意，修改中需要将奥斯卡图片挪到单独的一列中。另外还需要给#content 和#sidebar 外面包裹一层类名为.row 的 div。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport"  content="width=device-width,initial-scale=1.0" />
<title>And the winner isn't…</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,
    screen.height)+'; path=/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
<link href="css/custom.css" rel="stylesheet" type="text/css" />

</head>

<body>

<div id="wrapper" class="container">
  <!-- the header and navigation -->
  <div id="header" class="row">
    <div id="logo" class="col_12">And the winner is<span>n't...</span></div>
    <div id="navigation" class="row">
```

```
        <ul>
          <li><a href="#">Why?</a></li>
          <li><a href="#">Synopsis</a></li>
          <li><a href="#">Stills/Photos</a></li>
          <li><a href="#">Videos/clips</a></li>
          <li><a href="#">Quotes</a></li>
          <li><a href="#">Quiz</a></li>
        </ul>
      </div>
    </div>
    <div class="row">
      <!-- the content -->
      <div id="content" class="col_9 alpha omega">
        <img class="oscarMain col_3" src="img/oscar.png" alt="atwi_oscar" />
        <div class="col_6 omega">
        <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
        <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst
           the real cinematic heroes lose out. Not very Hollywood is it?</p>
        <p>We're here to put things right. </p>
      <a href="#">these should have won &raquo;</a>
      </div>
      </div>
      <!-- the sidebar -->
      <div id="sidebar" class="col_3">
        <div class="sideBlock unSung">
          <h4>Unsung heroes...</h4>
          <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" /></a>
          <a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt
              Earp" /> </a>
        </div>
        <div class="sideBlock overHyped">
          <h4>Overhyped nonsense...</h4>
          <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" /></a>
          <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
        </div>
      </div>
    </div>
    <!-- the footer -->
    <div id="footer" class="row">
      <p>Note: our opinion is absolutely correct. You are wrong, even if you think
          you are right. That's a fact. Deal with it.</p>
    </div>

  </div>
  </body>
  </html>
```

还需要给 `custom.css` 文件中追加一些额外的 CSS 样式，该文件的内容如下：

```
#navigation ul li {
  display: inline-block;
}

#content {
```

```
  float: right;
}

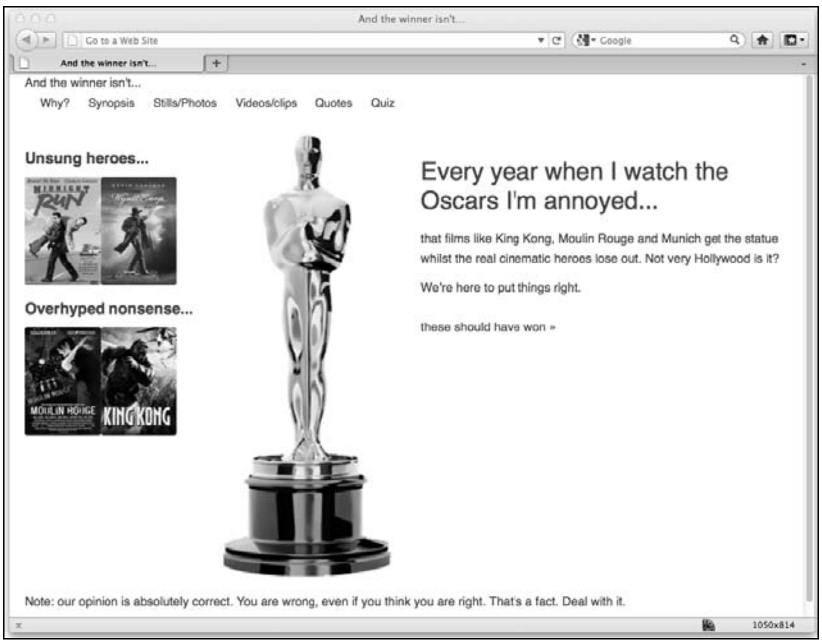
#sidebar {
  float: left;
}

.sideBlock {
  width: 100%;
}

.sideBlock img {
  max-width: 45%;
  float:left;
}

.footer {
  float: left;
}
```


在这些基本修改完成之后，在浏览器窗口中就能看到页面的基本结构已经搭好，且页面能随着浏览器视口的改变而缩放。



很明显还有很多细节工作要做（我知道，这样说太轻描淡写了），但如果你需要一种快速搭建响应式网站结构的方法，那么 Columnal 这样的 CSS 网格系统值得考虑。

3.9 小结

本章，我们学习了如何将死板的像素布局的网页结构改造成灵活的百分比布局的结构，学习了如何使用 em 来替代 px 创建更加灵活的文字版式，掌握了如何部署服务器端解决方案为不同尺寸的设备屏幕提供不同的图片，从而保证图片的响应性和流畅缩放。最后，我们试用了一套响应式 CSS 网格系统，它能让我们以最快的速度搭建起响应式网页结构。不过到目前为止，在我们追求响应式设计梦想的过程中，一直使用的是 HTML 4.01 标签。第 1 章我们提到过 HTML5 带给我们的一些好处。这些好处对响应式设计来说尤为重要和有意义，响应式设计的“移动优先”（mobile first）思想使它很适宜采纳最简洁、最有效和最具语义的代码。下一章，我们将认真学习 HTML5，并修改页面标签以便发挥最新最酷的 HTML 规范的优势。

响应式设计中的 HTML5 4

HTML5 是从 Web Applications 1.0 项目发展而来的，由网页超文本技术工作小组（WHATWG）发起，后被 W3C 采纳。HTML5 规范的大部分内容都偏重于处理 Web 应用程序。就算你不创建 Web 应用程序，也并不意味着 HTML5 对你制作响应式设计就没有太大价值。所以，HTML5 的一部分特性与制作更好的响应式网页直接相关（如简洁的代码），而另一些则和响应式设计没太大关系。

HTML5 还提供了针对表单处理和用户输入的特定工具。这些特性大大节省了为表单验证这类工作而在复杂技术如 JavaScript 上耗费的精力。不过我们将在第 8 章再单独研究 HTML5 表单。

本章内容

- HTML5 的那些部分我们现在就能用？
- 如何编写 HTML5 网页
- HTML5 的精简之道
- HTML 中的废弃零件
- HTML5 中的全新语义化元素
- 使用无障碍网页应用技术（WAI-ARIA）来增强语义，支持辅助技术
- 嵌入媒体
- 可响应的 HTML5 和 iFrame 视频
- 让网站支持离线使用

4.1 HTML5 的哪些部分现在就能用

虽然完整的 HTML5 规范尚待批准，但是现代浏览器都不同程度地支持 HTML5 的大多数新特性，这些浏览器包括 Safari、Google Chrome、Opera、Mozilla Firefox，而且还有 IE9！当前 HTML 标准草案的内容最终未必全都会出现在 W3C 推荐标准中，但其中有很大一部分新特性现在已经可以用了。

4.1.1　大多数网站可以用 HTML5 编写

现在，当我接到制作网站的任务，选择的默认标签会是 HTML5 而不是 HTML 4.01。和几年前的情况完全相反，现在，你得给网站不使用 HTML5 标签一个说得过去的理由。所有现代浏览器都能够正确理解常见的 HTML5 新特性（新的结构元素、视频和音频标签），而对老版本的 IE 则可以使用腻子脚本来弥补我所遇到过的所有缺陷。

> **什么是腻子脚本？**
>
> 腻子脚本（polyfill）这个词由 Remy Sharp 提出，意指使用腻子来补平老版本浏览器的缺陷。因此，腻子脚本具体指的是一段能给老版本浏览器带来新特性的 JavaScript 代码。值得注意的是，腻子脚本会给你的代码里追加多余的代码。因此，就算你添加 3 个腻子脚本可以让 Internet Explorer 6 中网站的渲染效果与其他浏览器一模一样，也并不意味着你一定要这么做！

4.1.2　腻子脚本和 Modernizr

通常，老版本的 Internet Explorer（IE9 以前的版本）并不识别 HTML5 的任何新语义元素。但是不久前，Sjoerd Visscher 发现如果先使用 JavaScript 创建这些元素，那 Internet Explorer 就能识别这些元素并可为其设置样式。基于这一发现，JavaScript 专家 Remy Sharp 开发了一个轻量级的增强脚本（http://remysharp.com/2009/01/07/html5-enabling-script/），在 HTML5 网页中引入该文件后，就能神奇地让老版本 IE 支持新 HTML 元素。长期以来，HTML5 的先驱们都会在页面中嵌入该脚本，以便让使用 IE 6、7 的用户享受和现代浏览器一样的体验。

不过此事现在有了更重大的进展。这个领域上出现了一个新丁，它能做到比刚才说的更多的事情。它的名字叫 Modernizr（http://www.modernizr.com），如果你正在编写 HTML5 页面，它很值得你去关注。除了能让 IE 支持 HTML5 新元素之外，它还能够基于一系列特性测试来有条件地加载更高级的腻子脚本（polyfill）、CSS 文件以及额外的 JavaScript 文件。

几乎没有什么理由不使用 HTML5，所以我们将开始写一点 HTML5 风格的页面。

> **想找一种编写优秀 HTML5 代码的捷径？可以考虑 HTML5 样板文件**
>
> 如果你时间紧迫，但却需要一个好的项目起点，可以考虑使用 HTML5 样板文件（http://html5boilerplate.com/）。样板文件是一个预先做好的融合了“最佳实践”HTML5 文件，包含一些基本样式（如之前提到过的 normalize.css）、polyfill 和一些必要的工具如 Modernizr。它还包含一个自动合并 CSS 和 JS 文件、自动删除注释以生成生产环境代码的构建工具。强烈推荐！

4.2　如何编写 HTML5 网页

打开一个已有的网页。你可能会看到文件开头有这样几行代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

删除上面的代码片段并将其替换成如下代码片段：

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
```

保存刚才的文档，现在你就有了自己的第一个 W3C 验证器（http://validator.w3.org/）认可的 HTML5 网页。

不要担心，本章尚未结束！刚才这个简单的练习只是为了展示 HTML5 的适应性。这只是已有代码的一次演进，而不是彻底革命。我们可以用它来增强我们已经掌握的标签代码。

到底该怎么做呢？首先，从新的 HTML5 文档类型声明开始：

```
<!DOCTYPE html>
```

如果你喜欢用小写字母，`<!doctype html>`也没问题。大小写没有区别。

> **HTML5 文档类型为什么这么简短？**
>
> HTML5 的<!DOCTYPE html>文档类型如此简短，目的是以简洁的方式告诉浏览器用“标准模式”渲染页面。这种简洁高效的思想渗透在 HTML5 的方方面面。

在文档类型声明之后，开写 HTML 标签，设置 `language` 属性，然后开写部分：

```
<html lang="en">
<head>
```

> **你会说汉语吗？**
>
> 根据 W3C 的定义（http://dev.w3.org/html5/spec/Overview.html#attr-lang），lang 属性用来指定页面元素内容和元素属性值的主语言。如果你的网页不是英语内容，你最好设定正确的页面语言。例如对于中文网页，HTML 标签应该是。完整的语言列表请见这个网址：http://www.iana.org/assignments/language-subtag-registry。

最后，设置一下字符编码。`meta` 是一个单闭合元素所以不需要关闭标签：

```
<meta charset=utf-8>
```

字符编码通常都是 UTF-8，除非你有特殊理由才可能使用其他编码。

4.2.1　HTML5 的精简之道

我记得在学校的时候，对我们超级严厉（其实是对我们负责）的数学老师最后都会因教不下去而走人。整个班级里充斥着被解放的愉悦，没有人因此对“严厉先生”（用这个名字来保护那些无辜的老师）感到惋惜。新来的老师一般都很和蔼可亲，他静静地坐在讲台上，任由我们胡闹也不会训斥或者给我们找麻烦。他没有强制要求我们在学习时要保持安静，也不关心我们的作业题解得如何精巧——他只关心答案。如果 HTML5 是一位数学老师，那它一定是那个好好先生。接下来我将证明这个奇怪的比喻……

如果你很注意代码风格，一般都会使用小写字母，将属性值用引号括起来，而且会为脚本或样式文件链接声明 type 属性。例如，你会使用如下的代码来引入一个样式表：

```
<link href="CSS/main.css" rel="stylesheet" type="text/css" />
```

HTML5 中不需要这么仔细，这样写它一样乐于接受：

```
<link href=CSS/main.css rel=stylesheet >
```

你看了一定感到怪异，我懂我也明白。代码没有闭合标签的斜线（/），属性值没有用引号括起来，没有 type 声明。但是，好说话的 HTML5 不在意这些。第二行示例代码和第一行一样有效。

这种宽松的语法可应用于整个文档，而不仅仅是链接 CSS 和 JavaScript 文件的元素。例如定义一个如下所示的 div：

```
<div id=wrapper>
```

这是完全有效合法的 HTML5 代码。插入一个图片：

```
<img SRC=frontCarousel.png aLt=frontCarousel>
```

这样同样有效。没有结束标签的斜线，没有引号，大小写混杂。甚至，省略<head>元素，页面依然有效。XHTML 1.0 对这种情况会怎么说呢？

4.2.2　HTML5 标签的合理写法

虽然我们的目标是采纳移动优先的响应式设计和开发思想，但我承认无法完全按照我所认为的最佳方式来编写页面（注意，我所举的例子都遵循基于 XML 语法的 XHTML 1.0 规范）。使用这种精简的编码方式确实可以节省极其微量的数据，但老实说，如果有这个需要，我宁愿通过从网页中删去一张图片来达到同样的目的。

对我而言，那些看似多余的字符（用于闭合的斜线和属性值两边的引号）可以提高代码的可读性。因此在编写 HTML5 文档时我倾向于在老式编写风格（这样风格的代码在 HTML5 中依然有效，不过可能在验证器或类似检查工具中会产生一些警告）和 HTML5 的极简主义之间找到一个平衡。举例说明，以上节的 CSS 链接为例，我会这样写：

```
<link href="CSS/main.css" rel="stylesheet"/>
```

我闭合了标签，使用了引号，但删除了 `type` 属性。关键是找到一种你自己满意的风格。HTML5 不会训斥你，也不会在课堂上批评你糟糕的标签代码，更不会因为你的代码没通过验证而罚你站墙角。

4.2.3　伟大的<a>标签万岁

HTML5 中一个真正便利的精简之处，是我们可以在标签中嵌入多个元素。（哇哈哈！终于来了。）以前，如果你想让代码通过验证，必须将每个元素单独使用标签来包裹。例如以下代码片段：

```
<h2><a href="index.html">The home page</a></h2>
<p><a href="index.html">This paragraph also links to the home page</a></p>
<a href="index.html"><img src="home-image.png" alt="home-slice" /></a>
```

现在，我们可以摆脱这个限制，像下面代码那样将一组元素包裹其中：

```
<a href="index.html">
<h2>The home page</h2>
<p>This paragraph also links to the home page</p>
<img src="home-image.png" alt="home-slice" />
</a>
```

唯一需要记住的是——很明显：不能在一个标签中嵌套另一个标签，也不能在标签中嵌套表单。

4.2.4　HTML 的废弃零件

除了 script 链接中的 language 属性，还有很多你可能已经习以为常的 HTML 标签或属性，在 HTML5 中都被认为是“废弃的”。HTML5 中的废弃零件分为两类——暂保留和非保留的。暂保留零件仍可正常运行但验证工具会报警告。在实际开发中尽量避免使用，但用了也不会让天塌下来。非保留的零件在某些浏览器中仍可以渲染，但如果你使用了它们，别人会认为你是顽冥不化的，周末恐怕谁都不愿意跟你玩。

举个使用暂保留的废弃零件的例子，即给图片设置 `border` 属性。这种方法以前用来去掉超链接中图片的蓝色边框。示例代码如下：

```
<img src="frontCarousel.png" alt="frontCarousel" border="0" />
```

现在则建议你使用 CSS 来完成同样的效果。

至于非保留的废弃零件都有哪些，多了去了。坦白地说，很多我从来都没用过（有些甚

至见都没见过！）。你的情况可能和我差不多。如果你好奇，可以在后面这个网址找到完整的非保留废弃零件列表：http://dev.w3.org/html5/spec/Overview.html#non-conforming-features。比较常见的非保留废弃零件有 strike、enter、font、acronym、frame 和 frameset。

4.3　HTML5 的全新语义化元素

字典中对语义学的定义是“关注语言本质含义的语言学和逻辑学分支学科”。对我们来说，语义化是给我们的标签赋予意义的过程。为什么语义化很重要？很高兴你问这个问题。看看 And the winner isn't...网站目前的标签结构：

```
<body>
<div id="wrapper">
  <div id="header">
    <div id="logo"></div>
    <div id="navigation">
      <ul>
        <li><a href="#">Why?</a></li>
      </ul>
    </div>
  </div>
  <!-- the content -->
  <div id="content">

  </div>
  <!-- the sidebar -->
  <div id="sidebar">

  </div>
  <!-- the footer -->
  <div id="footer">

  </div>
</div>
</body>
```

大多数网页开发者能看懂这些 div 上使用的通用的 ID 命名约定——header、content、sidebar 等等。但是对于代码本身，任何用户代理（网页浏览器、屏幕阅读器、搜索引擎爬虫等）看到这段代码都无法确定每个 div 的意义。HTML5 旨在使用全新的语义化元素来解决这个问题。接下来将从页面结构的角度来讲解这些元素。

4.3.1　<section>

<section>元素用来定义文档或应用程序中的区域（或节）。例如，可以用它组织你的个人信息，一个<section>用于联系信息，另一个用于新闻动态。需要重点理解的是用它的目的不是为了美化样式。如果你只想将某个元素包裹起来以便设置样式，那应该和以前一样继续使用<div>。

想知道 W3C 的 HTML5 标准对<section>的说明，请访问如下网址：http://dev.w3.org/html5/spec/Overview.html#the-section-element。

4.3.2 <nav>

<nav>元素用来定义文档的主导航区域，其中的链接指向其他页面或当前页面的某些区域。因为<nav>用于主导航区域，所以严格来讲它不是为页脚或其他经常会包含一组链接的区块而设计的（虽然将它用在这些区块里包含链接也没问题）。

想知道 W3C 的 HTML5 标准对<nav>的说明，请访问如下网址：http://dev.w3. org/html5/spec/Overview.html#the-nav-element。

4.3.3 <article>

<article>元素与<section>元素很容易混淆。在完全理解之前我只得一遍又一遍地阅读它们的标准定义。<article>元素用来包裹独立的内容片段。当搭建一个页面时，想想你准备放入<article>标签的内容能否作为一个整块而被复制粘贴到另外一个完全不同的网站且能保持完整的意义？另一种办法是，想想包裹在<article>中的内容能否在 RSS 订阅源中独立成为一篇文章？应该使用<article>标签包裹的内容，最明显的例子就是博客正文。注意，如果出现嵌套的<article>元素，那内层的<article>元素内容应该和外层文章内容直接有关。

想知道 W3C 的 HTML5 标准对<article>的说明，请访问如下网址：http://dev.w3.org/html5/spec/Overview.html#the-article-element。

4.3.4 <aside>

<aside>元素用来表示与页面主内容松散相关的内容。在实践中，我经常将其用作侧边栏（当它包含合适的内容时）。另外，引文、广告以及导航元素（如友情链接等）也可以使用它。

想知道 W3C 的 HTML5 标准对<aside>的说明，请访问如下网址：http://dev.w3.org/html5/spec/Overview.html#the-aside-element。

4.3.5 <hgroup>

如果页面中有一组使用<h1>、<h2>、<h3>等标签的标题、标语和副标题，则可以考虑使用<hgroup>将它们包裹起来。这样在 HTML5 的大纲结构算法中就会隐藏次级标题元素，从而只让<hgroup>中的第一个标题元素进入文档大纲。

HTML5 的大纲结构算法

HTML5 允许每个<section>容器有自己独立的大纲结构。这样你就不必总想着现在是几级标题了。例如在一个博客中，博文的标题可以使用<h1>标签，同时博客内容的标题也可以包含<h1>标签。请见如下代码：

```
<hgroup>
  <h1>Ben's blog</h1>
  <h2>All about what I do</h2>
</hgroup>
  <article>
    <header>
      <hgroup>
        <h1>A post about something</h1>
        <h2>Trust me this is a great read</h2>
        <h3>No, not really</h3>
        <p>See. Told you.</p>
      </hgroup>
    </header>
  </article>
```

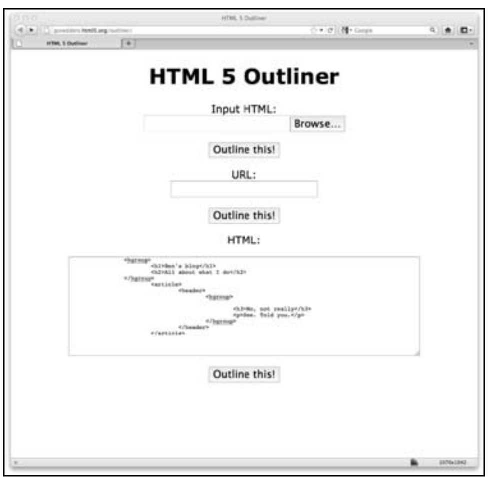
尽管代码中有多个<h1>和<h2>元素，但大纲结构还是这样：

- Ben’s blog
    - A post about something

因此，你就不必吃力地去记住标题级别了，而是可以在每个容器中随意使用任何级别的标题元素，HTML5 的大纲结构算法令自动组织好大纲。可以使用下面的 HTML5 大纲生成器来测试生成网页的大纲结构：

- ❑ http://gsneders.html5.org/outliner/
- ❑ http://hoyois.github.com/html5outlier/

下图展示了 HTML5 大纲生成器的界面：



> 想知道 W3C 的 HTML5 标准对<hgroup>的说明，请访问如下网址：http://dev.w3.org/html5/spec/Overview.html#the-hgroup-element。

4.3.6 <header>

由于<header>元素不计入大纲结构，所以不能用它来划分内容结构，而是应该用它来包含对区域内容的介绍说明。实际使用中，<header>可用作网站头部的“刊头”区域，也可用作对其他内容如<article>元素的简要介绍。

> 想知道 W3C 的 HTML5 标准对<header>的说明，请访问如下网址：http://dev.w3.org/html5/spec/Overview.html#the-header-element。

4.3.7 <footer>

和<header>一样，<footer>元素也不计入大纲结构，所以也不能用于划分内容结构。应该用它来包含其所在区域的辅助信息。例如可以用它包含一组指向其他页面的超链接，或者用它包含版权信息。和<header>一样，它也可以视情况在同一个页面上多次出现。例如博客网站的页脚可以用它，同时博客正文<article>元素内的文脚也可以用它。不过规范指出，博文作者的联系信息应该使用<address>元素来包裹。

> 想知道 W3C 的 HTML5 标准对<footer>的说明，请访问如下网址：http://dev.w3.org/html5/spec/Overview.html#the-footer-element。

4.3.8 <address>

<address>元素用于明确地标注离其最近的<article>或<body>祖先元素的联系信息。为避免产生混淆，请记住<address>中一般不放具体的邮政地址，除非相应内容确实需要联系地址。而邮政地址和其他可能会改变的联系信息应该使用<p>标签来包裹。

> 想知道 W3C 的 HTML5 标准对<address>的说明，请访问如下网址：http://dev.w3.org/html5/spec/Overview.html#the-address-element。

4.4 HTML5 结构元素的实际用法

我们来看几个使用刚才所讲的这些新元素的实际例子。我想<header>、<nav>和<footer>元素的用途是显而易见的，所以先从它们开始。从 And the winner isn’t...网站的首页开始，修改其头部、导航和页脚（注意看下面代码片段中加粗的部分）：

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
<meta name="viewport"  content="width=device-width,initial-scale=1.0" />
<title>And the winner isn't...</title>
<script>document.cookie='resolution='+Math.max(screen.width,screen.height)+';
path=/';</script>
<link href="css/main.css" rel="stylesheet" />

</head>

<body>

<div id="wrapper">
  <!-- the header and navigation -->
  <header>
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav>
      <ul>
        <li><a href="#">Why?</a></li>
        <li><a href="#">Synopsis</a></li>
        <li><a href="#">Stills/Photos</a></li>
        <li><a href="#">Videos/clips</a></li>
        <li><a href="#">Quotes</a></li>
        <li><a href="#">Quiz</a></li>
      </ul>
    </nav>
  </header>
  <!-- the content -->
  <div id="content">
    <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
    <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
    <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the
        real cinematic heroes lose out. Not very Hollywood is it?</p>
    <p>We're here to put things right. </p>
  <a href="#">these should have won &raquo;</a>
  </div>
  <!-- the sidebar -->
  <div id="sidebar">
    <div class="sideBlock unSung">
      <h4>Unsung heroes...</h4>
      <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" /></a>
      <a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt Earp"
          /></a>
    </div>
    <div class="sideBlock overHyped">
      <h4>Overhyped nonsense...</h4>
      <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" /></a>
      <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
    </div>
  </div>
  <!-- the footer -->
  <footer>
```

```
    <p>Note: our opinion is absolutely correct. You are wrong, even if you think
        you are right. That's a fact. Deal with it.</p>
  </footer>

</div>
</body>
</html>
```

不过，当页面中有`<article>`或`<section>`元素时，则刚才所说的这三个元素在每个页面中的使用次数不受限制。每个`<article>`或`<section>`元素都可以有自己的头部、脚注和导航。例如，我们在刚才的页面中添加一个`<article>`元素，代码如下：

```
<body>

<div id="wrapper">
  <!-- the header and navigation -->
  <header>
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav>
      <ul>
        <li><a href="#">Why?</a></li>
      </ul>
    </nav>
  </header>
  <!-- the content -->
  <div id="content">
    <article>
      <header>An article about HTML5</header>
      <nav>
        <a href="1.html">related link 1</a>
        <a href="2.html">related link 2</a>
      </nav>
      <p>here is the content of the article</p>
      <footer>This was an article by Ben Frain</footer>
    </article>
```

如上代码所示，我们在页面及其所包含的`<article>`中都使用了`<header>`、`<nav>`和`<footer>`元素。

接下来继续修改侧边栏区域，其当前的 HTML 4.01 标签结构如下：

```
<!-- the sidebar -->
  <div id="sidebar">
    <div class="sideBlock unSung">
      <h4>Unsung heroes...</h4>
      <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" /></a>
      <a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt
          Earp" /></a>
    </div>
    <div class="sideBlock overHyped">
      <h4>Overhyped nonsense...</h4>
      <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" /></a>
      <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
    </div>
  </div>
```

4

侧边栏内容当然是和页面主要内容“略微”相关的，所以首先移除`<div id="sidebar">`并将其替换为`<aside>`：

```
<!-- the sidebar -->
  <aside>
    <div class="sideBlock unSung">
      <h4>Unsung heroes...</h4>
      <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" /></a>
      <a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt
          Earp" /></a>
    </div>
    <div class="sideBlock overHyped">
      <h4>Overhyped nonsense...</h4>
      <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" /></a>
      <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
    </div>
  </aside>
```

代码看上去很优雅！但是如果在浏览器中看到效果之后你张口骂娘也是情有可原的……



这真是进一步退两步！造成这种糟糕效果的原因是我们没有针对新元素修改 CSS 代码。在我们继续前进之前得先把这件事搞定。我们得将所有`#header` 引用改为 `header`，所有`#navigation` 引用改为 `nav`，所有`#footer` 引用改为 `footer`。例如，第一个引用`#header` 的 CSS 规则如下：

```
#header {
  background-position: 0 top;
  background-repeat: repeat-x;
  background-image: url(../img/buntingSlice3Invert.png);
  margin-right: 1.0416667%; /* 10 ÷ 960 */
  margin-left: 1.0416667%; /* 10 ÷ 960 */
  width: 97.9166667%; /* 940 ÷ 960 */
}
```

要将其改为：

```
header {
  background-position: 0 top;
  background-repeat: repeat-x;
  background-image: url(../img/buntingSlice3Invert.png);
  margin-right: 1.0416667%; /* 10 ÷ 960 */
  margin-left: 1.0416667%; /* 10 ÷ 960 */
  width: 97.9166667%; /* 940 ÷ 960 */
}
```


这些修改非常简单，因为 header、navigation 和 footer 作为 ID 与其对应要修改的元素名称一致——我们只需要去掉“#”即可。sidebar 则有些许不同：我们得将带`#sidebar` 的引用改为 `aside`。不过，在编辑器中执行“查找与替换”可帮助我们快速完成任务。简要说一下，类似这样的 CSS 规则：

```
#sidebar {
}
```

要变成这样：

```
aside {
}
```

即使你写了一个超长的 CSS 样式文件，将其中的 HTML 4.01 ID 名称替换成 HTML5 元素也不是什么难事。

> **注意 HTML5 中的元素复用**
>
> 请注意，HTML5 页面中可能会有多个`<header>`、`<footer>`和`<aside>`元素，你可能需要为每一个都编写特定的样式。

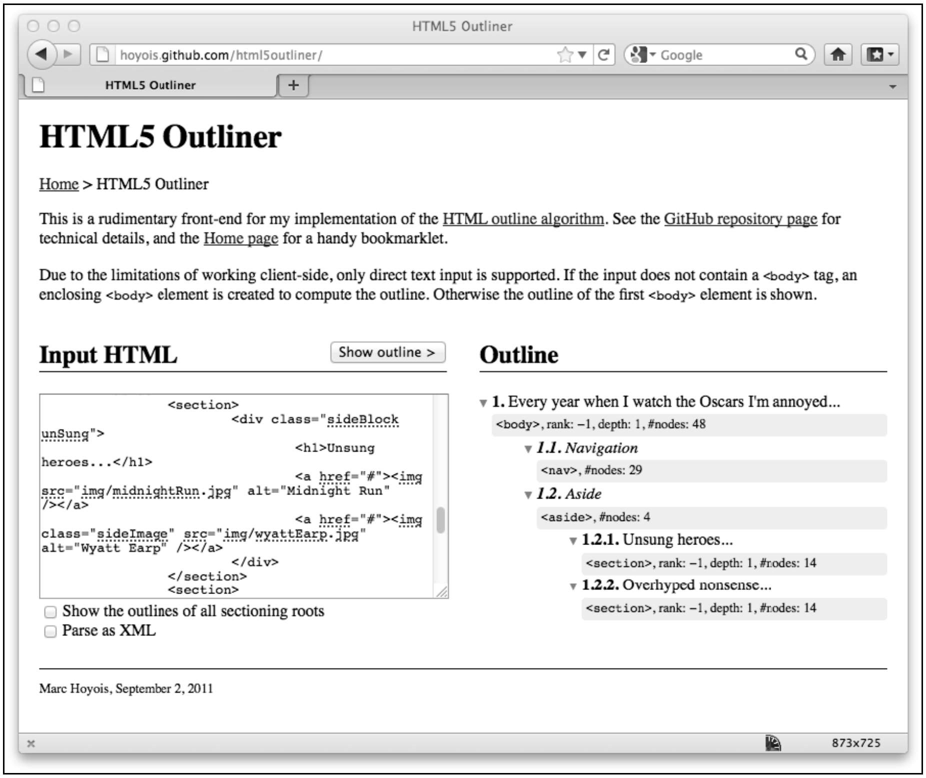
And the winner isn’t...的样式修改完成后，我们就又回到了正轨：



现在，我们虽然告知了用户代理（如浏览器）页面那部分是侧边栏，但其中又含有两个独立的区域，UNSUNG HEROES 和 OVERHYPED NONSENSE。为了让这两个区域具有语义，还要对代码做进一步修改：

```
<!-- the sidebar -->
  <aside>
    <section>
      <div class="sideBlock unSung">
        <h4>Unsung heroes...</h4>
        <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" /></a>
        <a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt
            Earp" /></a>
      </div>
    </section>
    <section>
      <div class="sideBlock overHyped">
        <h4>Overhyped nonsense...</h4>
        <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" /></a>
        <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
      </div>
    </section>
  </aside>
```

需要谨记的是，使用`<section>`的目的不是为了美化样式，而是为了标识一个鲜明独立的内容块。一个内容块（section）一般都应该带有标题，这恰好符合我们的需求（用于独立标识）。考虑到 HTML5 的大纲结构，我们可以将上面代码中的`<h4>`标签改为`<h1>`，这样 HTML5 就会为文档生成一个精确的大纲：



网站的主体内容怎么办

你可能会觉得有点奇怪：没有一个专门的元素用来标记页面主体内容。其实逻辑是这样的，既然可以界定除主体内容之外的其他元素，那么剩下的元素自然就是页面的主体了。

4.5　HTML5 的文本级语义元素

除了前面讲过的结构元素，HTML5 还修订了一些被称之为行内元素的标签。HTML5 标准中称这些标签为文本级语义元素（http://dev.w3.org/html5/spec/Overview.html#text-level-semantics）。我们来看几个常用的例子。

4.5.1　<b>

过去，人们通常利用<b>元素为文本添加样式，但它的实际用途其实是“给文本加粗”。不过现在你可以正式地将其仅用作样式钩子了，因为现在 HTML5 标准对<b>的定义是：

> ……一小段文本，纯粹为了吸引人的注意，除此之外不传达任何重要性，也不暗示其他语态或语气，如文档摘要中的关键词、评论中的产品名称、交互式文本软件中的可操作单词，或者文章的导语。

4.5.2　<em>

OK，我举手坦白，我也常利用<em>为文本添加样式。不过现在我需要改进使用方法，因为在 HTML5 中它的语义是：

> ……强调内容中的重点。

因此，除非你确实想强调标签中的内容，否则的话可以考虑使用<b>标签或者可以的话使用<i>标签。

4.5.3　<i>

HTML5 标准中对<i>的描述如下：

> ……一小段有不同语态或语气的文字，或者是样子上与普通文章有所差异以便标明不同特点的文字。

简单地说，它不仅仅是用来给某些文字加斜体效果的。

4.5.4　在页面中应用文本层语义元素

来看看网站首页主要内容部分的标签，看它能不能给用户代理提供更好地语义。当前的标签结构如下：

```
<!-- the content -->
  <div id="content">
    <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
    <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
    <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst
        the real cinematic heroes lose out. Not very Hollywood is it?</p>
    <p>We're here to put things right. </p>
    <a href="#">these should have won &raquo;</a>
  </div>
```

确实可以对这些代码做些改善。首先，<h1>标签中的<span>标签在上下文中毫无语义，

而我们想要在样式中为其设置强调效果，所以在 HTML 代码中也应该如此：

```
<h1>Every year <em>when I watch the Oscars I'm annoyed...</em></h1>
```

回顾一下网站之前的效果：



还需要给电影名称设置不同的样式，但它们不需要带有不同的语气或语态。看来在此处使用<b>标签很合适：

```
<p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and <b>Munich</b> get the
statue whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>
```

> **文本层语义元素的默认样式**
>
> 根据过去对<b>标签的用法，很多浏览器仍会将其渲染为粗体。所以你可以根据实际情况在相关的 CSS 代码中重定义其默认样式。

最后，当我说“‘we’re here to put things right”时，我的意思是我不是瞎胡闹的，而且我想让用户代理（浏览器）知道这点。所以最后，我们将这句话使用<i>标签包裹。你可能会争论说此处应该使用<em>标签，<em>其实也可以，但我准备使用<i>。修改后的代码如下：

```
<p><i>We're here to put things right.</i></p>
```

和<b>元素一样，浏览器默认会将<i>中的文本渲染为斜体，如果需要，就重定义其默认样式吧。

现在我们给网页主要内容中添加了一些文本级语义元素。HTML5 还有大量文本级语义标签，想要全面了解，请查看 HTML5 标准中的相关章节，地址如下：

http://dev.w3.org/html5/spec/Overview.html#text-level-semantics

其实，再付出一点点努力，使用辅助技术为用户提供额外的语义，可以使我们的语义化工作更上一层楼。

4.6　遵循 WAI-ARIA 实现无障碍站点

WAI-ARIA 是 Web Accessibility Initiative - Accessible Rich Internet Applications 的缩写，指无障碍网页应用技术，它主要解决一个问题：让残障人士能无障碍地访问网页上的动态内容。这种技术提供了一种描述自定义组件（网页应用中的动态片段）的角色、状态和属性的方法，这样这些组件就可以被依赖辅助技术的用户找到并加以利用。

例如，屏幕上的一个组件正在显示不断变化的股票价格，如何让访问网页的盲人用户也知道这一点呢？无障碍网页应用技术就在尝试解决这类问题。本书着眼点不在于全面讲解无障碍网页应用技术（想要全面了解，请查看 http://www.w3.org/WAI/intro/aria），但我们可以采纳其中一部分容易实施的技术，将其应用到 HTML5 网站中，以方便残障用户。

如果你接了一个为客户制作网站的任务，除了基本要求之外通常不会给你专门的时间/经费来增加无障碍支持（悲剧的是，无障碍性经常被完全抛诸脑后）。但我们还是可以使用无障碍网页应用技术中的地标角色（landmark role）来修正 HTML 语义元素的一些明显的不足，从而使支持无障碍网页应用技术的屏幕阅读器可以在不同的页面区块之间轻松跳转。

ARIA 的地标角色

在响应式网页设计中实现 ARIA 的地标角色，跟响应式设计没有什么关系。可是，在某种程度上支持它一下又非常简单（而且不必作任何修改就保证通过 HTML5 验证），假如为了省那么点事儿，就对从今往后编写的所有 HTML5 页面持放任态度，似乎也真不值得。好了，不贫了，来看看怎么做吧。

新的 HTML5 导航区域结构如下：

```
<nav>
  <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
```

```
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
  </ul>
</nav>
```

我们可以让导航区域在支持无障碍网页应用技术的屏幕阅读器上轻松地定位，只需为其追加一个地标角色属性 role 即可，如下面的代码片段所示：

```
<nav role="navigation">
  <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
  </ul>
</nav>
```


够简单吧？针对文档结构的各部分分别有如下的地标角色。

- ❑ `application`：用来定义用作网页应用的区域。
- ❑ `banner`：用来定义一个站点级别（而不是某个特定文档的）的区域。如网站的头部和 logo。
- ❑ `complementary`：用来定义一个对页面主要区域进行补充说明的区域。在 And the winner isn't...这个网站中，UNSUNG HEROES 和 OVERHYPED NONSENSE 区域就可以定义为 `complementary`。
- ❑ `contentinfo`：用来定义与页面主要内容相关的信息区域。例如页脚的网站版权信息区域。
- ❑ `form`：你猜都能猜到，定义表单！但注意，如果表单用于搜索，则请使用 `search` 来替代。
- ❑ `main`：定义页面的主体内容。
- ❑ `navigation`：用来定义链向当前文档或相关文档的导航链接。
- ❑ `search`：用来定义一个用于搜索的区域。

> **无障碍网页应用技术进阶**
>
> 无障碍网页应用技术并非只有地标角色。想要做进阶应用，请参阅完整的角色列表及其简要使用说明：http://www.w3.org/TR/wai-aria/roles#role_definitions。

让我们从头开始，将现在 HTML5 版的 And the winner isn't...网站标签使用地标角色加以扩展：

```
<body>
<div id="wrapper">
  <!-- the header and navigation -->
  <header role="banner">
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav role="navigation">
      <ul>
        <li><a href="#">Why?</a></li>
        <li><a href="#">Synopsis</a></li>
        <li><a href="#">Stills/Photos</a></li>
        <li><a href="#">Videos/clips</a></li>
        <li><a href="#">Quotes</a></li>
        <li><a href="#">Quiz</a></li>
      </ul>
    </nav>
  </header>
  <!-- the content -->
  <div id="content" role="main">
    <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
    <h1>Every year <em>when I watch the Oscars I'm annoyed…</em></h1>
    <p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and <b>Munich</b>
        get the statue whilst the real cinematic heroes lose out. Not very
        Hollywood is it?</p>
    <p><i>We're here to put things right.</i></p>
        <a href="#">these should have won &raquo;</a>
  </div>
  <!-- the sidebar -->
  <aside>
    <section role="complementary">
      <div class="sideBlock unSung">
        <h1>Unsung heroes...</h1>
        <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" /></a>
        <a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt
            Earp" /></a>
      </div>
    </section>
    <section role="complementary">
      <div class="sideBlock overHyped">
        <h1>Overhyped nonsense...</h1>
        <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" /></a>
        <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
      </div>
    </section>
  </aside>
  <!-- the footer -->
  <footer role="contentinfo">
    <p>Note: our opinion is absolutely correct. You are wrong, even if you think
        you are right. That's a fact. Deal with it.</p>
  </footer>

</div>
</body>
```

使用非可视桌面阅读器（NVDA）免费测试网站可访问性

如果你是在 Windows 平台上开发且想使用屏幕阅读器测试网站的可访问性，可以免费使用 NVDA。软件的官方地址如下：http://www.nvda-project.org/。

希望本节对无障碍网页应用技术的简要介绍，让你看到了为使用辅助技术的用户提供一定可访问性支持还是很简单的，希望你能在下一个 HTML5 项目中考虑一下。

为使用了无障碍技术的元素设置样式

和其他属性一样，可以直接使用属性选择器来为其设置样式。如，使用 `nav[role="navigation"] {}`可以为导航区域设置样式。

4.7　在 HTML5 中嵌入媒体

对很多人来说，HTML5 首次进入他们的视线是在苹果公司拒绝在 iOS 设备上支持 Flash 的时候。Flash 作为浏览器视频服务插件已经获得了市场主导（有人说是市场垄断）地位。但是，苹果公司没有使用 Adobe 的专有技术，而是决定依靠 HTML5 来处理富媒体渲染。HTML5 本身在这一领域已经有了长足进步，苹果公司对 HTML5 的公开支持又极大促进了它的发展，并使其媒体工具在公众中赢得了更广泛的青睐。

正如你所预想，IE8 及更低版本都不支持 HTML5 视频和音频。不过给微软的"先天不足"的浏览器提供备用解决方案也很简单，稍后我们会讨论这个问题。其他现代浏览器（Firefox 3.5+、Chrome 4+、Safari 4、Opera 10.5+、Internet Explorer 9+、iOS 3.2+、Opera Mobile 11+、Android 2.3+）都没问题。

4.8　用 HTML5 的方法为页面添加视频或音频

说实话，我发现在 HTML 4.01 网页中添加视频或音频媒体完全是一种折磨。不是因为它很难，而是很麻烦。HTML5 使这件事情变简单了。添加多媒体的语法和添加图片类似：

```
<video src="myVideo.ogg"></video>
```

对很多网页设计师来说简直如沐春风！不再需要向网页中引入一大堆代码，在 HTML5 中只需要一个`<video></video>`标签（音频使用`<audio></audio>`标签）就能搞定这种吃力活。还可以在开始标签和结束标签之间插入文字用以告知那些使用不兼容 HTML5 浏览器的用户，此外还支持你一般都会追加的附加属性如 `height` 和 `width`。我们把这些都加上：

```
<video src="video/myVideo.mp4" width="640" height="480">What, do you mean you don't
understand HTML5?</video>
```

现在将上面这段代码插入我们的网页然后在 Safari 中查看，视频就会出现但没有播放控制栏。想要显示默认的播放控制栏则需要追加 controls 属性。我们还可以追加 autoplay 属性（不建议——因为大家都讨厌自动播放的视频，这是常识）。修改后的代码如下：

```
<video src="video/myVideo.mp4" width="640" height="480" controls autoplay>What, do
you mean you don't understand HTML5?</video>
```

上面的代码所产生的效果如下图所示：



其余的属性还包括用来控制媒体预加载的 preload（HTML5 的早期尝鲜者应该注意 preload 替代了原先的 autobuffer），用来重复播放视频的 loop，以及用来定义视频缩略图的 poster（这个属性在视频播放延迟时非常有用）。要使用某个属性，将其追加到标签中即可。下面的例子包含了刚才提到的所有属性：

```
<video src="video/myVideo.mp4" width="640" height="480" controls autoplay
preload="auto" loop poster="myVideoPoster.jpg">What, do you mean you don't
understand HTML5?</video>
```

4.8.1　提供备用的媒体源文件

最初的 HTML5 规范呼吁所有浏览器内置支持使用 Ogg 格式[1]直接播放视频或音频（无需插件）。但是由于 HTML5 工作组的内部争议，曾经作为基线标准的支持 Ogg（包括 Theora video 和 Vorbis audio）的主张在最近更新的 HTML5 规范中被放弃。因此目前的情况是，一些浏览器支持某一套视频和音频文件格式，而另一些浏览器则支持其他格式。例如 Safari 只允许在<video>和<audio>元素中使用 MP4/H.264/AAC 媒体文件，而 Firefox 和 Opera 则只支持 Ogg 和 WebM。

> 为什么我们不能和睦相处！（《火星人玩转地球》，http://movie.douban.com/subject/1297715/。）

谢天谢地，有一种方法能在一个标签内支持多种媒体格式。但是这种方法并不能免除我们为一个媒体文件创建多种版本。我们都期望这个问题在将来某个适当的时刻会自行解决，此时我们手握多种格式的媒体文件，则可以这样编写视频标签：

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
    <source src="video/myVideo.ogv" type="video/ogg">
    <source src="video/myVideo.mp4" type="video/mp4">
    What, do you mean you don't understand HTML5?
</video>
```

如果浏览器支持 Ogg 格式，则使用第一个文件；否则它会继续往下解析下一个<source>标签。

4

4.8.2　针对老版本浏览器的备用方案

照这种方式使用<source>标签，我们就能根据需要提供一系列备用方案。例如在提供了 MP4 和 Ogg 格式之后，如果我们还想给 IE8 及更低版本提供一个优雅的备用方案，则可以追加一个 Flash。更进一步，如果用户的浏览器没有任何合适的媒体播放技术，我们还可以为其提供媒体文件的下载链接：

```
<video width="640" height="480" controls autoplay preload="auto" loop poster=
    "myVideoPoster.jpg">
    <source src="video/myVideo.mp4" type="video/mp4">
    <source src="video/myVideo.ogv" type="video/ogg">
    <object width="640" height="480" type="application/x-shockwave-flash"
        data="myFlashVideo.SWF">
    <param name="movie" value="myFlashVideo.swf" />
    <param name="flashvars" value="controlbar=over&amp;image=myVideoPoster.
        jpg&amp; file=video/myVideo.mp4" />
    <img src="myVideoPoster.jpg" width="640" height="480" alt="__TITLE__"
```

① 详见 http://baike.baidu.com/view/99249.htm。——译者注

```
        title="No video playback capabilities, please download the video below" />
    </object>
    <p>  <b>Download Video:</b>
  MP4 Format:  <a href="myVideo.mp4">"MP4"</a>
  Ogg Format:  <a href="myVideo.ogv">"Ogg"</a>
    </p>
</video>
```

4.8.3 和标签的用法基本一致

<audio>标签的用法与<video>基本一致，除了 width、height 和 poster 之外其他属性基本相同。你甚至可以将<video>和<audio>标签互换使用。两者之间的最大差别就是<audio>没有可视内容的播放区域。

4.9 响应式视频

以上我们可以看出，支持老版本浏览器会一如既往地导致代码变得臃肿。<video>标签从刚开始的一两行变成到最后的十多行（增加了一个 Flash 文件），仅仅是为了让老版本浏览器舒坦工作！对我而言，我更乐意忘掉 Flash 备用方案以追求更精简的代码，不过每种用法都各有优劣。

现在，我们可爱的 HTML5 视频的唯一问题是它不是响应式的。没错，我们辛辛苦苦搞的响应式网页却不怎么响应。看看下面的截图，你最好忍着不要飙泪：



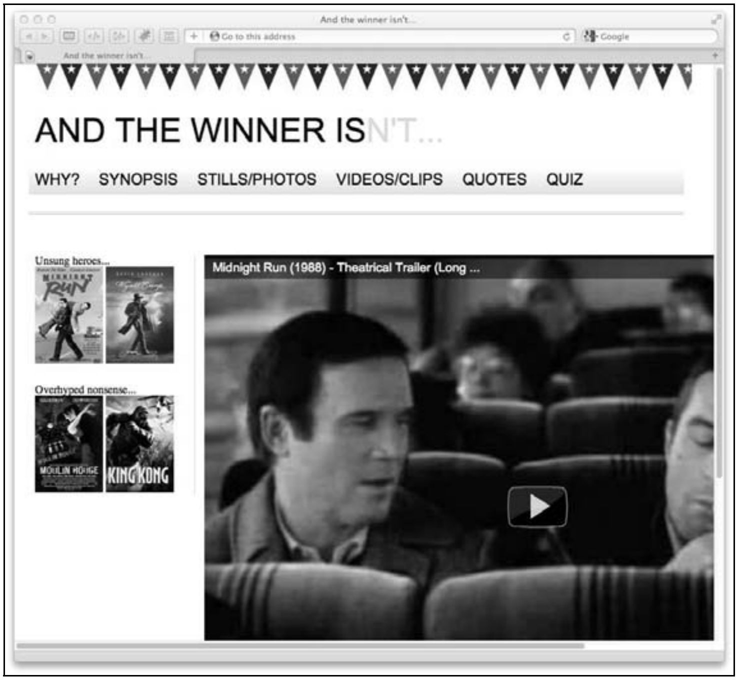
幸运的是，对于 HTML5 式嵌入视频，修正方法很简单。只需删除视频标签中的 height 和 width 属性（如删除 width="640" height="480"），然后在 CSS 中追加如下代码：

```
video { max-width: 100%; height: auto; }
```

这种方法对本页面中的视频文件很有用，但它不能解决使用 iframe 嵌入的视频的响应问题（拜 YouTube、Vimeo 等等视频网站所赐）。下面的代码可以在网页中插入来自 YouTube 的电影《午夜狂奔》[1]的预告片：

```
<iframe width="960" height="720" src="http://www.youtube.com/embed/B1_N28DA3gY"
frameborder="0" allowfullscreen></iframe>
```

暂且不管之前的 CSS 规则，现在的效果如下：



4

我敢肯定德尼罗[2]对这个效果很不爽！很有多方法可以解决这个问题，但截至目前我见过的最简单的办法是使用一个名为 FitVids 的 jQuery 小插件。我们来看看将这个插件应用到 And the winner isn't...网站中到底有多简单。

首先引入 jQuery 库文件。在页面的<head>元素中加载该文件。此处我使用来自 Google

①《午夜狂奔》：http://movie.douban.com/subject/1294699/。——译者注
② 德尼罗：《午夜狂奔》的主演，截图中右边那位先生。——译者注

的内容分发网络（CDN）的文件。

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js">
</script>
```

其次，从网站 http://fitvidsjs.com/上下载 FitVids 插件（有关该插件的更多信息请见 http://daverupert.com/2011/09/responsive-video-embeds-with-fitvids/）。

将 FitVids 文件存入一个合理的文件夹（假设文件夹名为“js”），然后在<head>中引入该文件：

```
<script src="js/fitvids.js"></script>
```

最后，只需使用 jQuery 指定包含 YouTube 视频的特定元素。本例中我将《午夜狂奔》视频放入了 id 为#content 的 div 中：

```
<script>
  $(document).ready(function(){
    // Target your .container, .wrapper, .post, etc.
    $("#content").fitVids();
  });
</script>
```

只需这三步。多亏有了 FitVid 插件，现在我们有了一个完全可响应的 YouTube 视频。（注意：小子，不要向视频中的德尼罗学习，吸烟有害健康！）



哈哈，全部搞定。这下我就又可以收到鲍比的圣诞贺卡了！

4.10　离线 Web 应用

虽然 HTML5 中的大量新特性对我们的响应式设计没有明显帮助（如地理定位 API），但离线 Web 应用应该还有点用处。我们知道肯定会有越来越多的移动设备用户访问我们的网站，为他们提供一种不需要网络连接仍可访问网站内容的途径如何？HTML5 的离线 Web 应用特性将其变成了可能。

这个功能很明显是用于 Web 应用的（真够搞笑的，不知道他们对这个名字是怎么想的）。假设有一个在线笔记应用，当用户的手机网络断开时，他可能正在编辑一则笔记。使用 HTML5 的离线 Web 应用，他就可以继续离线编辑笔记，然后等到网络再次连接时将本地数据发送到服务器。

HTML5 离线 Web 应用的最美妙之处在于它的设置和使用都超级简单。接下来，我们将使用这项技术——为我们的网站创建一个离线版本。这意味着如果用户想在没有网络的情况下访问我们的网站，他也能做到！

4.10.1　离线 Web 应用概述

离线Web应用的运行机制是每个需要离线使用的网页都指定一个后缀名为`.manifest`的文本文件。这个文本文件罗列了该网页离线使用时所需的所有资源文件（HTML、图片 JavaScript 等等）。支持离线 Web 应用的浏览器会自动读取`.manifest` 文件，下载文件中所罗列的资源文件，并将其缓存在本地以备网络断开时使用。简单吧？那我们来动手试一试……

4.10.2　让网页可离线使用

在 HTML 的开始标签中，我们指定一个.manifest 文件：

```
<html lang="en" manifest="/offline.manifest"
```

该文件的文件名随意，但后缀名建议使用`.manifest`。

> 你必须在每一个准备离线使用的页面的HTML标签中都追加`manifest="/offline.manifest"`属性。

如果使用的是 Apache 服务器，你可能还需要修改一下`.htaccess` 文件，追加一行代码：

```
AddType text/cache-manifest .manifest
```

这样就保证了`.manifest` 文件拥有正确的 MIME 类型，即 `text/cache-manifest`。

在`.htaccess` 文件中还可以加入以下代码：

```
<Files offline.manifest>
  ExpiresActive On
  ExpiresDefault "access"
</Files>
```

添加上面这几行代码，可以阻止浏览器缓存缓存文件。你没看错。因为 offline.manifest 是一个静态文件，浏览器默认就会缓存 offline.manifest 文件。所以上面这几行代码就是让服务器告诉浏览器不要这么干！

现在我们需要给 offline.manifest 填充内容。即通知浏览器那些文件是用作离线存储的。And the winner isn't...网站的 offline.manifest 文件内容如下：

```
CACHE MANIFEST
#v1

CACHE:
basic_page_layout_ch4.html
css/main.css
img/atwiNavBg.png
img/kingHong.jpg
img/midnightRun.jpg
img/moulinRouge.jpg
img/oscar.png
img/wyattEarp.jpg
img/buntingSlice3Invert.png
img/buntingSlice3.png

NETWORK:
*

FALLBACK:
//offline.html
```

4.10.3　理解 manifest 文件

manifest 文件必须以 CACHE MANIFEST 开头。第二行就是一句注释，注明了 manifest 文件的版本号。这句注释的用途稍后详细介绍。

CACHE:部分罗列了所有离线使用所需的文件。这些文件的路径都是相对 offline.manifest 而言的，所以文件路径可能需要根据情况稍作修改。使用绝对路径也是可以的。

NETWORK:部分罗列了所有不需要被缓存的文件。你可以将其看成是一个“在线白名单”。此处罗列的文件在网络畅通的情况下都会直接跳过缓存。如果你想网站内容在网络畅通的情况下及时更新（而不是仅在离线缓存中查找），可以在此处使用*。星号被称为在线白名单通配符。

FALLBACK:部分使用/字符定义了一个 URL 模板。它的作用是访问每个页面时都会问“缓存中有这个页面吗？”，如果有则显示缓存页面，如果没有则显示指定的 offline.html 文件。

4.10.4 页面被自动加载到离线缓存

根据实际情况，还有一种更简单的办法来设置 `offline.manifest` 文件。任何指定了离线 manifest 文件的页面（就是在标签中追加了 `manifest="/offline.manifest"`的页面）在被用户访问时都会被自动加入到本地缓存。浏览器会缓存用户访问过的每一个网页以确保这些网页在离线状态下仍可访问。简化的 manifest 文件如下：

```
CACHE MANIFEST
# Cache Manifest v1
FALLBACK:
//offline.html
NETWORK:
*
```

选择使用这个方法时有一点需要注意，这种方法只会下载和缓存用户访问的 HTML 页面，不会缓存页面内引入的图片、JavaScript 或者其他资源文件。如果这些资源文件是必需的，那么请按照上节中的方法在 `CACHE:`部分专门声明这类文件。

4

4.10.5 版本注释的用途

对网站内容或任何资源文件做了修改之后，你必须得对 `offline.manifest` 文件也做点修改并将其重新上传服务器。这样就能让服务器为浏览器提供新版本文件，而浏览器则会下载该新版本文件并再次触发离线存储进程。我效仿了 Mark Pilgrim 的例子（来自著名的《畅游 HTML5》一书），在 `offline.manifest` 文件的头部加了一句注释，每次修改网站都会对应地修改该版本号：[①]

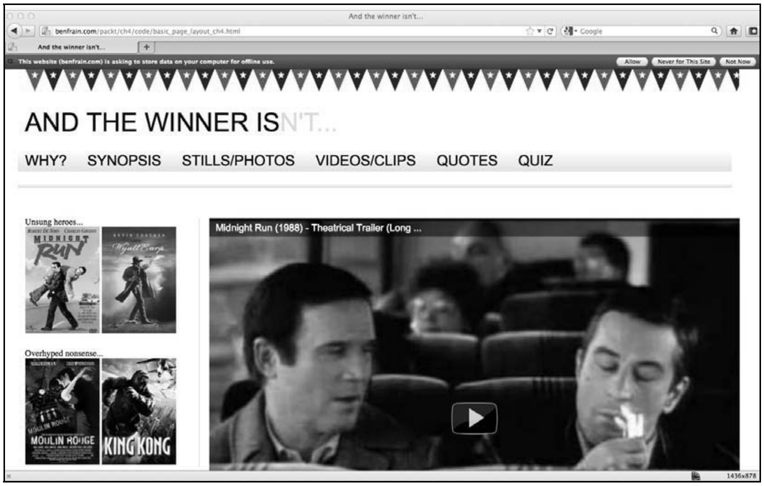
```
# Cache Manifest v1
```

4.10.6 离线访问网站

现在来测试一下我们的作品。在支持离线 Web 应用的浏览器中访问网页。一些浏览器会提示该网页使用了离线模式并在本地保存数据（例如 Firefox，注意看下面截图中顶部的通知栏），Chrome 则不会提示：

---

① 作者此处说得不是很清楚。补充一下：如果开发者对网站内容或资源做了修改，那么也得通知浏览器更新缓存文件，否则浏览器仍然会使用之前已有的缓存文件。而通知浏览器更新缓存文件的方式通常是更新 manifest 文件，浏览器如果发现 manifest 文件发生了变化，就会更新缓存文件。大多数情况下 manifest 中的缓存文件清单不会发生变化，那我们就通过修改注释的方式来改变 manifest 文件，注释中的版本号，既能触发文件变化，又能指明当前版本，一举两得。其实注释中还可以加入更新时间等更详细信息，有助于维护。文中提到的《畅游 HTML5》，参见 http://diveintohtml5.com/。——译者注



接着，拔掉网线（或者关闭 WiFi——这个没有“拔掉网线”听着霸气）然后刷新浏览器。网页应该会和联网状态下一样刷新显示——其实我们没联网。

4.10.7　离线 Web 应用的故障诊断

当网站在离线状态下出现问题导致无法正常运行时，我一般都使用 Chrome 来做故障诊断。Chrome 内置的开发人员工具有一个非常好用的 Console 控制台（点击地址栏右侧的扳手图标，然后找到**工具 | 开发者工具**，之后切换到 Console 面板；或者直接按 F12），从中可以看出哪些文件缓存成功，那些缓存失败，以及你做错了什么。以我的经验，通常是文件路径出现问题，例如缓存页面相对于 manifest 文件的路径不正确。



离线 Web 应用的完整规范，请见如下网址：

http://dev.w3.org/html5/spec/Overview.html#offline

4.11　小结

这一章讲了很多内容。从最基本的使用 HTML5 制作页面，到确保用户断网时仍可访问网页的离线应用技术。我们还学习了在标签中嵌入富媒体（视频）以及如何确保它在不同视口中可响应，学习了如何编写更富语义的代码，以及如何帮助那些依赖辅助技术的用户——尽管这些技术并不只适用于响应式设计。不过我们的网站仍然有一些明显的缺陷。毫不客气地说，网站非常简陋：文字没有样式，页面元素缺少细节，对照一下设计图中的按钮效果就知道了。我们不会用传统的加载图片方式来解决这些问题，理由是我们不需要图片！在下一章，我们将会使用强大而灵活的 CSS3 来制作更加快速和更易维护的响应式设计。

CSS3：选择器、字体和颜色模式

5

第 1 章我们提到过，通过移动通信网络来浏览网页的人越来越多。当前电信网络的速度差异很大，所以我们需要考虑带宽限制和网站的加载时间。这就像回到了当年使用 56K 调制解调器的时代，不得不考虑页面及其中包含的图片和多媒体文件需要加载多长时间。现在，我们得面对与之类似的加载时间问题。正如表格布局技术中的百分比规则重新流行一样，现在有必要重新检查插入页面中的每一个媒体片段和耗费带宽的内容。尽管我们的设备已普遍移动化，但下载内容的速度和下载所产生的费用（速度和费用）仍和几年前类似。但如今旧貌却换了新颜！CSS3 可以大幅减少视觉效果对图片的依赖，为我们制作可在极短时间内加载的漂亮网站提供了有力工具。CSS3 有很多内容要讲。第 6 章会讲解更多 CSS3 技巧，包括文字阴影、盒阴影、渐变和背景图片；同时第 7 章会学习 CSS3 动画、变形和转换。

本章内容

- CSS3 给前端开发人员带来了什么
- 快速而便捷的 CSS3 技巧（多列布局和文字换行）
- CSS 规则解析
- 私有前缀的来源和用法
- 新的 CSS3 选择器的工作原理
- 使用`@font-face` 设定字体
- 如何使用带有透明度的 RGB 和 HSL 颜色模式

5.1 CSS3 给前端开发人员带来了什么

过去，我们要么押注用户愿意为优雅漂亮的设计多等点时间（顺便说一句，用户不愿意），要么为了可用性而抛弃图片，牺牲设计理念。CSS3 让我们可以在很多方面不必再妥协和牺牲。仅需要几行代码（而且不用图片），CSS3 就可以创造出各种效果：圆角、背景渐变、

文字阴影、盒阴影、自定义字体以及多重背景图片（当然，这个效果确实需要图片）。如果这还不够神奇，甚至我们之前需要依赖 JavaScript 的一些基本交互效果如悬停动画，也可以使用纯 CSS3 来实现。CSS3 所蕴含的海量利好及精简之道，可以让我们将响应式设计从“一个普通的可响应网站”提升为一个面向未来的真正响应式网站。使用 CSS3 之后，可以让响应式设计加载更快，所需资源更少，且在将来更容易维护和修改。在进入正题之前，我们先来解决一个棘手的问题。

5.1.1 Internet Explorer 6 到 8 对 CSS3 的支持

除少数例外（如`@font-face`），老版本的 IE（IE 6、7、8）几乎不支持 CSS3 的新特性。那么我们可以在设计开发中使用 CSS3 吗？和以前一样，答案依然是：“看情况。”

就我个人而言，目前我主要将 CSS3 用作增强网站，而不是用它提供基本功能。我非常喜欢页面元素在不同的浏览器中看起来有点差异，相信你和你的客户也是这样。你会发现回顾一下 1.11 节“引导客户：网站不必在所有浏览器中表现一致”会很有帮助。评价网站某部分“可行”或“看上去不错”是很主观的。但是你得知道有很多腻子脚本（polyfill）可以为老版本 IE 增加 CSS3 功能。想要应用这些腻子脚本，应该详细参阅第 9 章的内容。

> 各版本 Internet Explorer 对 CSS 2.1 和 CSS3 特性的支持情况，请见如下网址：http://msdn.microsoft.com/en-us/library/cc351024%28v=vs.85%29.aspx（或 http://tinyurl.com/495756c）。

5.1.2 使用 CSS3 设计和开发页面

我不能替你说话，但我确实发现重做图片很烦人。你知道我说的就是这些意见——“能不能让圆角再平滑点？”或者“渐变的顶部颜色能不能再深点？”一旦我们老老实实地做了修改，则不可避免的会听到：“噢，算了，刚才那样挺好。你能改回去吗？”现在当然可以，这种来来回回的过程不可避免；毕竟我们常常只想改改设计看看效果。CSS3 可以让你使用代码在几秒钟之内搞定这些，而不用在图片编辑器里花费好几分钟。

5.2 CSS 规则解析

在探索 CSS3 带给我们的新特性之前，为避免混淆，我们先来定义一下用来描述 CSS 规则的一些术语。看看如下的例子：

```
.round {
  border-radius: 10px;
}
```

这条规则由选择器（`.round`）和声明（`border-radius: 10px;`）组成。而声明则由属性（`border-radius:`）和值（`10px;`）组成。我们的说法一致吧？很好，继续。

5.3 私有前缀及其用法

在 CSS3 模块标准尚未被 W3C 批准或者标准所提议的特性尚未被浏览器完全实现时，浏览器厂商会使用所谓的**私有前缀**来测试“试验性的”CSS 特性。这样，浏览器设计者实现了 CSS3 的新模块，但对使用 CSS3 的开发者来说则很繁琐。看看 CSS3 中实现圆角的代码：

```
.round{
  -khtml-border-radius: 10px; /* Konqueror */
  -rim-border-radius: 10px; /* RIM */
  -ms-border-radius: 10px; /* Microsoft */
  -o-border-radius: 10px; /* Opera */
  -moz-border-radius: 10px; /* Mozilla (如Firefox) */
  -webkit-border-radius: 10px; /* Webkit (如Safari和Chrome) */
  border-radius: 10px; /* W3C */
}
```

你可以看到一组私有前缀属性（这个列表还不是很全面），每一个都有其独有的前缀字符串，如`-webkit-`是针对 Webkit 核心浏览器，`-ms-`是微软的私有前缀所以针对的是 Internet Explorer，等等。CSS 的工作方式是浏览器逐行下载样式表，应用其可识别的属性，忽略其无法识别的属性。

此外，样式表中后出现的属性优先级高于之前出现的同名属性。正是由于这种层叠，我们就可以先列出私有前缀属性，最后使用无前缀的属性来修正，以确保当该特性被完全实现时，浏览器会运行正确的效果，而不是之前的特定浏览器的试验性效果。

> **可快速编辑 CSS3 前缀的代码片段和 JavaScript 方案**
>
> 建立包含所有必需的私有前缀属性的代码片段会带来极大便利。这样你就可以直接粘贴代码而不用每次都重新输入。很多代码编辑程序（或者集成开发环境）带有代码片段功能，这样在开发 CSS3 时能节省很多时间。也有可以为 CSS 文件自动追加前缀的 JavaScript 方案，比如“-prefix-free”，很不错的解决方案，网址是 http://leaverou.github.com/prefixfree/。

挨个列出每一种私有前缀是最理想的做法，但实际开发中很少有人那样做。相反，他们要么只指定自己期望的浏览器，要么在编写规则之前已经检查过哪些浏览器支持该特性。例如，你可以仅写如下代码：

```
.round{
  -moz-border-radius: 10px; /* Mozilla (e.g Firefox) */
  -webkit-border-radius: 10px; /* Webkit (e.g. Safari and Chrome) */
```

```
  border-radius: 10px; /* W3C */
}
```

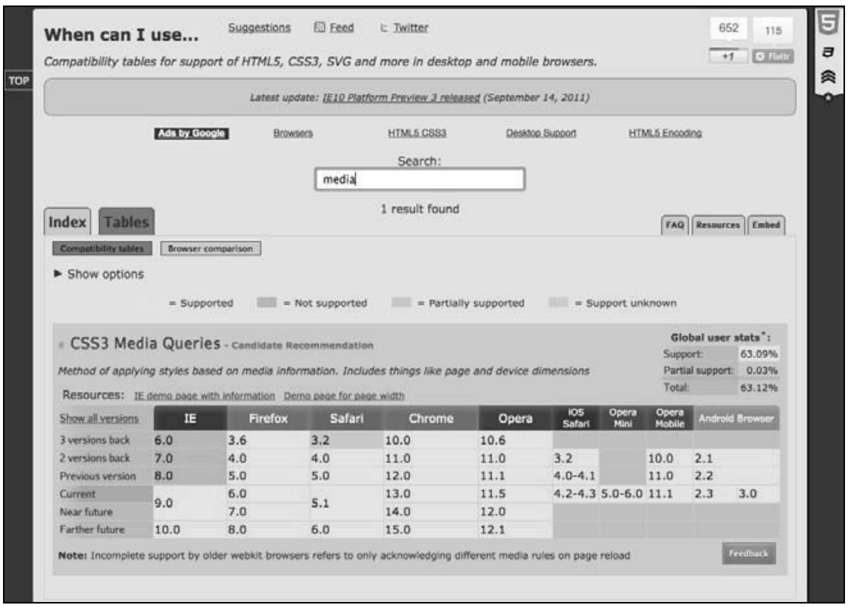
这样写会覆盖 Firefox、Chrome 和 Safari，以及任何已经完全实现了该规则的浏览器。

我知道你在想什么，对同一个属性写多个私有前缀声明是否会造成代码臃肿？确实会有那么一点。但无论追加多少前缀属性，相对于图片而言，这仍然是一个更加快速、优雅且健壮的解决方案。

在讨论网站应用之前，最好先看看当前浏览器使用率的分布情况。这样做会让我们对支持哪些浏览器更胸有成竹。例如，如果时间和预算都很紧张，你可能会决定不对任何在你的网站上使用率小于 3%的浏览器提供私有前缀支持。此时你需要根据一组数据来做判断。

现在我们知道了前缀的来源，以及如何在规则中应用它。接下来我们看一些快速而有效的 CSS3 技巧。

什么时候可以使用特定的 CSS3 和 HTML5 特性？



随着对 CSS3 研究的不断深入，我衷心建议大家去看看这个网站 http://caniuse.com，在这里你可以知道当前浏览器对特定 CSS3 和 HTML5 特性的支持程度。除了显示浏览器的特性支持情况之外（可按特性搜索），它还提供了来自于 http://gs.statcounter.com 的最近的全球浏览器使用率统计。

5.4　快速而有效的 CSS 技巧

在我的日常工作中，有一些 CSS3 新特性会被经常使用，还有一些则基本不用。在开始冗长的讲解之前，我想先给大家分享几个能让工作更轻松的 CSS3 小技巧，尤其是响应式设计方面的，用它们可以完成以前会让人头疼的简单任务。

5.4.1　CSS3 多栏布局

曾经有需要将一整段文本显示在多个栏位中？在 CSS3 出现之前，你必须将内容拆分到不同的标签中，然后分别设定样式。因为样式原因而修改标签永远不是什么好做法。CSS3 可以让我们将一段或多段内容分布到多列网格中。请看如下代码：

```
<div id="main" role="main">
    <p>lloremipsimLoremipsum dolor sit amet, consectetur
    // 任意文字 //
</p>
    <p>lloremipsimLoremipsum dolor sit amet, consectetur
    // 任意文字 //
</p>
</div>
```

你可以通过设定具体栏位宽度（如 `12em`）或者栏位数量（如 `3`）来使内容分布在多列网格中，做法如下。

如果设定栏位宽度，语法如下所示（注意，为简洁起见代码中省略了私有前缀）：

```
#main {
  column-width: 12em;
}
```

按照上述的设定，无论视口尺寸是多少，内容都会分布在宽度为 12em 的栏位中。视口尺寸发生变化之后，浏览器会自动调整栏位数量。

如在视口宽度 1024 像素的 Safari 浏览器中效果如下：



而下图则展示了相同页面在视口宽度为 768 像素的 iPad 上的显示效果：



只需极少量代码就能完成这样精美的响应式布局——我喜欢！

如果你想保持栏位数量不变而让栏位宽度根据视口自动调整，可以参考如下代码：

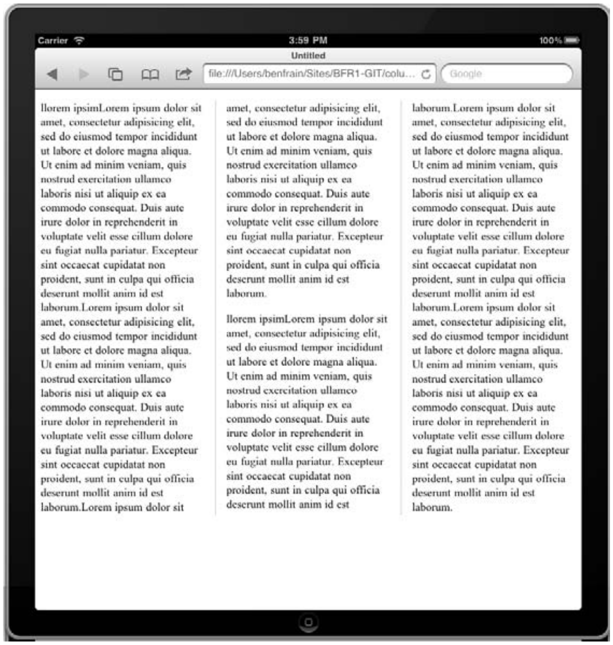
```
#main {
  column-count: 4;
}
```

增加栏位间隙和分割线

增加栏位间隙和分割线可以让多列布局的效果更好，代码如下：

```
#main {
  column-gap: 2em;
  column-rule: thin dotted #999;
  column-width: 12em;
}
```

效果如下图：

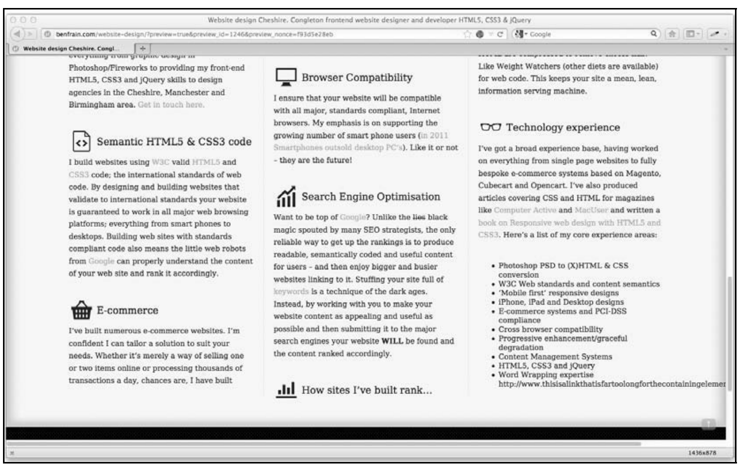


若想参阅 CSS3 多列布局模块的标准，请访问 http://www.w3.org/TR/css3-multicol/。

目前，请切记你需要给多列布局声明使用私有前缀，以确保兼容最广泛的浏览器。

5.4.2　文字换行

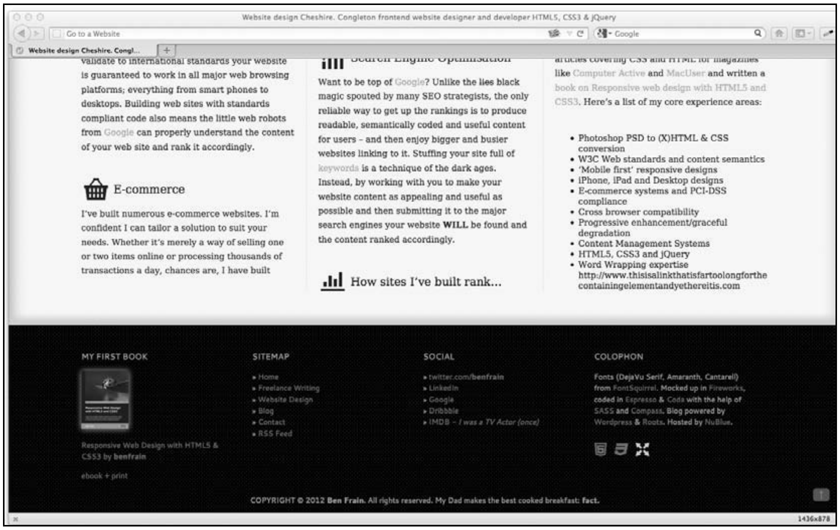
有多少次你必须得将一个很长的 URL 地址放置在一个狭小的空间内？结果很让你扫兴吧？请看下面截图中的问题，注意页面右下角的 URL 地址已经超出了它的范围：



CSS3 使用了一个简单的声明解决了这个问题，凑巧的是老版本 IE 均支持该声明，甚至可以追溯到 IE 5.5！

```
word-wrap: break-word;
```

给外层的包裹元素追加该声明后，产生的效果如下图所示。接下来是见证奇迹的时刻，超长的 URL 完美换行了！



5.5　CSS3 的新增选择器及其用法

CSS3 赋予了我们超级强大的能力，用以选择页面内的元素。你可能觉得这听起来没什么好激动的，但相信我，这种能力会让你的工作无比轻松，你会因此爱上 CSS3！我想，最好还是让事实来证明这个大胆的预言……

5.5.1　CSS3 属性选择器

你可能已经使用过现有的 CSS 属性选择器来指定规则了，如下代码所示：

```
img[alt] {
  border: 3px dashed #e15f5f;
}
```

这个选择器会匹配页面标签中任意一个含有 `alt` 属性的图片标签：

```
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
```

还可以通过设定属性值来缩小匹配范围。如下代码所示：

```
img[alt="atwi_oscar"] {
  border: 3px dashed #e15f5f;
}
```

这样就仅会匹配 alt 属性值为 atwi_oscar 的图片。刚才说的这些事 CSS2 都能搞定。CSS3 给我们带来了什么新特性？其实就是三种“子字符串匹配”的属性选择器。

1. CSS3 的子字符串匹配属性选择器

CSS3 可以让我们基于属性选择器的子字符串来选择元素。听起来有点复杂，实际上并不深奥。换句话说，现在我们可以根据属性的部分内容来选择元素。三种匹配模式分别是：

- 以特定前缀开头；
- 包含特定字符串；
- 以特定后缀结尾。

我们来看看具体用法。

(1)“匹配开头”的属性选择器。

“匹配开头”的属性选择器语法如下：

```
Element[attribute^="value"]
```

在实际使用中，如果我想选择网站中所有 alt 属性值以 film 开头的图片，则对应代码如下：

```
img[alt^="film"] {
  border: 3px dashed #e15f5f;
}
```

该选择器的关键字符是^符号，它的意思是“以此开头”。

(2)“匹配包含内容”的属性选择器。

“匹配包含内容”的属性选择器语法如下：

```
Element[attribute*="value"]
```

在实际使用中，如果我想选择网站中所有 alt 属性值中包含 film 字符串的图片，则对应代码如下：

```
img[alt*="film"] {
  border: 3px dashed #e15f5f;
}
```

该选择器的关键字符是*符号，它的意思是“包含”。

(3)“匹配结尾”的属性选择器。

“匹配结尾”的属性选择器语法如下：

```
Element[attribute$="value"]
```

在实际使用中，如果我想选择网站中所有 alt 属性值以 film 结尾的图片，则对应代码如下：

```
img[alt$="film"] {
  border: 3px dashed #e15f5f;
}
```

该选择器的关键字符是$，它的意思是“以此结尾”。

2. 一个活生生的例子

这些子字符串属性选择器如何在实际工作中发挥作用呢？举一个我经常使用的 CSS3 属性选择器的例子。假设我使用内容管理系统（如 WordPress、Concrete 或者 Magento）来制作网站，那网站肯定有让用户添加新页面的功能。用户可能会添加一些有关他们公司或新产品的新闻。他每在内容管理系统中添加一个新页面，生成的 HTML 代码中的<body>或相关标签就会包含一个特定的 ID 值，用以标识该页面，以便和其他页面区分开来。例如某个用户爱好赛车运动并在网站上维护着一个“赛事年历”版块。这样每一个<body>标签都会有一个与年份对应的 ID 值：

```
<body id="2003">
```

> **HTML5 中的 ID 值可以用数字开头**
>
> 如果你还不习惯 HTML5，可能会认为 ID 值以数字开头是无效的，因为 HTML 4.01 是这样规定的。HTML5 取消了这个限制，HTML 的 ID 命名方面唯一需要记住的是名字中间不能有空格且保证在页面中唯一。详细信息请参阅 http://dev.w3.org/html5/spec/Overview.html#the-id-attribute。

我想让指向“赛事年历”的导航栏链接在任意一个赛车年度页面时均被高亮显示，以表示它和“赛车年历”版块相关。我肯定不会写一个覆盖每一个未来年份的样式规则，而是写一个以不变应万变（也可以说是防患于未然）的 CSS3 规则：

```
body[id^="2"] .navHistory { color: #00b4ff; }
```

这个规则表示任意含有.navHistory 类名的元素，只要它被包含在 ID 值以 2 开头的<body>中，则其文字颜色为#00b4ff。一个简单的规则覆盖了所有可能性。当然除非这个网站在公元 3000 年时还保持目前的样子——那时，即使我吃嘛嘛香身体倍儿棒，估计也老得没法继续维护该网站了。

5.5.2　CSS3 结构伪类

做的网站越多，越会发现自己总是在一遍遍地解决同样的问题。我举个典型的例子：水平导航栏一般都是由一组相同间距的<li>链接组成。假设我们想让每一个导航链接——

但不包括第一个和最后一个，左右两边都有一定的外边距。以前的解决办法是给第一个和最后一个`<li>`元素上分别追加一个语义上多余的类名，如下面代码片段中加粗的代码行所示：

```
<ul>
  <li class="first"><a href="#">Why?</a></li>
  <li><a href="#">Synopsis</a></li>
  <li><a href="#">Stills/Photos</a></li>
  <li><a href="#">Videos/clips</a></li>
  <li><a href="#">Quotes</a></li>
  <li class="last"><a href="#">Quiz</a></li>
</ul>
```

然后在 CSS 中追加两个样式，并修改针对那两个特殊列表项的外边距值：

```
li {
  margin-left: 5%;
  margin-right: 5%;
}
.first {
  margin-left: 0px;
}
.last {
  margin-right: 0px;
}
```

这样是能解决问题，但不够灵活。比如当我们使用内容管理系统来制作网站时，系统可能会自动添加新的导航链接，此时再给代码中的列表项追加或删除 `last` 或 `first` 类以保证结构正确就不是件容易的事了。

1. :last-child 选择器

CSS2.1 中已经有一个针对列表中第一项的选择器：

```
li:first-child
```

CSS3 又增加了一个选择器用以匹配最后一项：

```
li:last-child
```

组合使用这两个选择器，就不需要在代码中增加额外的类名了。

我们将使用这个方法，并结合 `display: table` 属性来改进 And the winner isn't...网站的导航。下面的截图展示了网站目前的样子：


And the winner isn't...
AND THE WINNER ISN'T...
WHY?
SYNOPSIS
STILLS/PHOTOS
VIDEOS/CLIPS
QUOTES
QUIZ
Unsung heroes...
Overhyped nonsense...
EVERY YEAR WHEN I WATCH THE OSCARS I'M ANNOYED...
that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?
We're here to put things right.
these should have won »
NOTE: OUR OPINION IS ABSOLUTELY CORRECT. YOU ARE WRONG, EVEN IF YOU THINK YOU ARE RIGHT. THAT'S A FACT. DEAL WITH IT.

我们来看看视觉设计图：


AND THE WINNER ISN'T...
WHY?
SYNOPSIS
STILLS/PHOTOS
VIDEOS/CLIPS
QUOTES
QUIZ
UNSUNG HEROES...
full info
full info
OVERHYPED NONSENSE...
full info
full info
EVERY YEAR
WHEN I WATCH THE OSCARS I'M ANNOYED...
that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it? We're here to put things right.
THESE SHOULD HAVE WON
NOTE: OUR OPINION IS ABSOLUTELY CORRECT. YOU ARE WRONG, EVEN IF YOU THINK YOU ARE RIGHT. THAT'S A FACT. DEAL WITH IT.

效果图中的导航栏链接占据了整个网页宽度，我们得重现这个效果。导航栏的标签代码如下：

```
<nav role="navigation">
  <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
  </ul>
</nav>
```

首先，我们设定 `nav` 元素使用 `table` 布局：

```
nav {
  display: table;
  /* more code... */
}
```

然后将`<ul>`显示为 `table-row`：

```
nav ul {
  display: table-row;
  /* more code... */
}
```

最后将列表项显示为 `table-cells`：

```
nav ul li {
  display: table-cell;
  /* more code... */
}
```

这样做可以保证如果有另外的列表项追加进来，同样会自动地调整它们之间的间距。最后，使用 CSS3 选择器将最后一个列表项的文字置为右对齐，将第一个列表项的文字置为左对齐。

```
nav ul li:last-child {
  text-align: right;
}

nav ul li:first-child {
  text-align: left;
}
```

此时在浏览器中看看效果，导航栏接近原始设计图中的效果了：



不用担心，此处的 table 只是显示模式而已

你可能会奇怪我到底是怎么想的，怎么会建议对导航栏布局使用 `table` 模式。不用担心，这些 `table` 只是表象。也就是这些表格只会存在于 CSS 中，并不会对页面标签产生任何影响。我们只是告诉浏览器让这些元素表现得如同表格一样，但其实它们并不是表格。将标签显示为这种模式也不会阻碍我们为不同的视口使用不同的布局模式，例如你可以在视口小于 768 像素时使用 `display: inline-block`。

2. nth-child 选择器

原始设计图中的导航栏链接交替使用不同的文字颜色，这个怎么实现呢？CSS3 还提供了一个选择器，可以在不追加额外标签代码的情况下解决该问题：

```
:nth-child(even)
```

我们先用这个选择器来解决上述的问题，然后再看看几种 `nth-child` 的用法——可解决之前需要额外标签才能搞定的问题。要实现导航链接中交替的红色文字，请追加如下样式：

```
nav ul li:nth-child(even) a {
  color: #fe0208;
}
```

现在导航栏文字就有了交替颜色效果：



怎么样？不需要一行 jQuery 代码也不需要额外的标签！我说什么来着，CSS3 选择器很强大吧！

3. 理解 nth 规则的作用

那些没好好学数学的前端工程师和网页设计师，最容易被 nth **规则**吓得浑身发抖（当然，写 PHP 代码或者帮别人构造正则表达式除外）。那接下来我们就试试啃下这块“硬骨头”，让那些后端大神也对咱们刮目相看。

提及在 DOM（文档对象模型，或者说简单点就是页面标签中的元素）树形结构中选择元素，CSS3 提供了一些基于 nth 的规则，为我们带来了前所未有的灵活性。这些规则包

括:nth-child(n)、:nth-last-child(n)、:nth-of-type(n)以及:nth-last-of-type(n)。前面的例子中已经示范了使用(odd) 或 (even)参数（用以修正导航链接），(n)这个参数还可以有其他几种形式：

- 使用整数，如:nth-child(2)——这会选中列表中第二个列表项；
- 使用数值表达式，如:nth-child(3n+1)——这样会从第一个元素开始，然后每三个元素选一个。

整数值很好理解，就是你想选择的元素的序号。而数值表达式对我们来说有点不好理解，我们来分析一下。为了便于理解，我们从括号内表达式的右边开始分析。例如我想确定(2n+3)会选择那些元素，就从右边开始（即从第 3 个元素开始），然后就知道它是从这个元素开始每两个元素选择一个。为说明问题，可以将之前的导航链接规则修改成这样：

```
nav ul li:nth-child(2n+3) a {
  color: #fe0208;
}
```

可以看到，第 3 个链接是红色的，之后每两个就选择一个列表项显示为红色（如果有 100 个列表项，这个规则就会在此基础上继续下去）：



那如何选择第二个列表项之后的所有列表项呢？可以写成:nth-child(1n+2)，但其实不需要写第一个数字 1，因为 1 乘以 n 还是等于 n，所以可以简写为:nth-child(n+2)。与此类似，如果想选择序号为 3 的倍数的元素，就不用写成:nth-child(3n+3)，直接写:nth-child(3n)就行，因为 3 的倍数肯定是从第 3 个开始的，无需专门声明。

数值表达式中也可以使用负数，例如:nth-child(3n-2)，即表示从倒数第 2 个元素开始然后每三个元素选择一个。将我们的导航链接规则照此修改：

```
nav ul li:nth-child(3n-2) a {
  color: #fe0208;
}
```

浏览器中的效果如下：



谢天谢地，我讲明白了吧？

child 和 last-child 的区别在于，last-child 是从文档节点树的末尾开始算。比如:nth-last-child(-n+3)就是从倒数第 3 个元素开始，向后选择之后的所有元素（因为使用了-n，所以方向是向后）。使用该规则后浏览器中的效果如下图所示：



最后，我们来看看:nth-last-of-type。前面的例子中在对子元素计数时都未考虑元素的类型，:nth-last-of-type 则可以指定你想选择的元素类型。请看如下代码结构：

```
<ul>
  <li class="internal"><a href="#">Why?</a></li>
  <li><a href="#">Synopsis</a></li>
  <li class="internal"><a href="#">Stills/Photos</a></li>
  <li class="internal"><a href="#">Videos/clips</a></li>
  <li class="internal"><a href="#">Quotes</a></li>
  <li class="internal"><a href="#">Quiz</a></li>
</ul>
```

注意上面的第二个列表项没有 internal 类。

看看这个规则：

```
nav ul li.internal:nth-of-type(n+2) a {
  color: #fe0208;
}
```

上面的代码告诉浏览器："从第二个匹配元素开始，选择每一个类名为 `internal` 的列表项。"浏览器中的效果如下：



> **CSS3 的计数方式和 jQuery 不太一样**
>
> 如果你经常使用 jQuery 就会知道，jQuery 中的计数是从 0 开始的。比如在 jQuery 中使用整数 1 来选择元素，实际上会选中第二个元素。但在 CSS3 中，计数从 1 开始，所以整数 1 会匹配第一个元素。

4. 否定（:not）选择器

另一个便利的选择器是否定伪类选择器，用于选择不满足某些条件的元素。例如，继续使用前面例子中的结构代码，将规则修改为：

```
nav ul li:not(.internal) a {
  color: #fe0208;
}
```

可以看出我们是想选择没有 `internal` 类的列表项。浏览器中的效果如下所示：



至此我们已经学习了**结构伪类**（详细信息请见 http://www.w3.org/TR/selectors/#structural-

pseudos）的主要内容。除此之外，CSS3 还有很多其他选择器。如果你在开发 Web 应用，那完整的 UI 元素状态伪类列表（http://www.w3.org/TR/selectors/#UIstates）则值得一读，它对你极有帮助，例如可以让你根据元素被选中与否来应用样式规则。

5.5.3 对伪元素的修正

伪元素在 CSS2 中已经存在，CSS3 标准对其语法做了一些细微的修正。举几个你可能还有印象的例子，`p:first-line` 会选中<p>标签的第一行内容，`p:first-letter` 会选中其中的第一个字母。CSS3 要求对伪元素使用两个冒号以便与伪类进行区别。因此刚才的例子应该改写为 `p::first-letter`。但注意 Internet Explorer 8 及更低版本的 IE 无法识别两个冒号的语法，它们只识别一个冒号。

:first-line 对响应式设计来说好用吗

:first-line 伪元素非常方便的一个特点是它会根据视口自动变化。例如如下的规则：

```
p::first-line {
  color: #ff0cff;
}
```

如你所想，第一行文字被渲染为可怕的粉红色（这让我想起了《红磨坊》[①]）：



视口大小不同时，渲染为粉红色的文字片段也不相同：

① Moulin Rouge：movie.douban.com/subject/5073826/。——译者注



于是，在响应式设计中就有了一种方法，不需要修改标签代码，即可方便地将文本第一行内容（即浏览器渲染出来的第一行，不是标签代码中的第一行）显示得与众不同。

希望这几节对 CSS3 选择器的简短突击学习，让你明白了使用它们不必修改现有设计和增加新标记。过去我经常需要使用 JavaScript 库（如 jQuery）来做复杂的元素选择，现在 CSS3 基本上消灭了这种需求。还有一点令人欣慰的是，CSS3 选择器模块现在已处于推荐标准状态，也就是说它已是一个非常成熟的模块，从现在起不会再有大的改动。

5.6　自定义网页字体

多年来我们一直被迫使用一组单调乏味的 Web 安全字体。当网页设计中确实需要一些优雅的字体时，我们通常都是使用图片来替代，并对元素使用 `text-indent` 规则将实际的文本移出视口范围。

曾有一些稍微高级点的备选方案，来为网页增加更加个性的字体效果。sIFR（http://www.mikeindustries.com/blog/sifr/）和 Cufón（http://cufon.shoqolate.com/generate/）都使用 Flash 和 JavaScript 来重建文本元素，然后使用自定义字体来显示内容。但是在响应式设计中，我们力求精简、语义和内容优先，所以多余的图片和臃肿的代码应该尽可能避免。幸好，CSS3 提供了一种自定义网页字体的方法，用它可以描绘美丽的新时代。

5.6.1　@font-face 规则

`@font-face` 规则在 CSS2 中已经存在（但随后在 CSS 2.1 中被删除）。IE 4 甚至对其提供了部分支持（真的，不骗你）！那现在都 CSS3 了，怎么还提它呢？

因为它又回来了！`@font-face` 已经被重新引入 CSS3 字体模块（http://www.w3.org/TR/css3-fonts）。过去在网页中使用自定义字体一直很麻烦，直到最近才开始有了一些真正可行的网页字体解决方案。但浏览器厂商对不同的字体格式和具体实现仍在争论。如 Embedded OpenType（EOT）字体是 Internet Explorer 的首选格式（其他浏览器都不支持），其他浏览器更钟爱常见的 TrueType 格式（TTF），还有 Scalable Vector Graphics (SVG)以及 Web Open Font Format (WOFF)。使用`@font-face` 为网页设定字体时，可谓喜忧参半。我们先说忧吧……

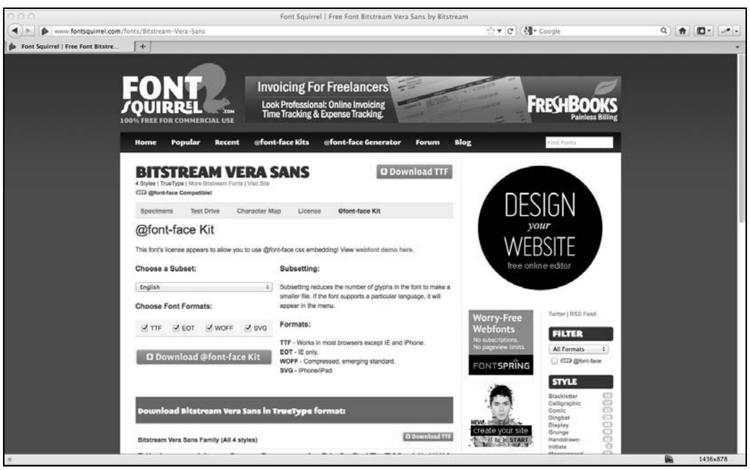
在某一个通用的字体格式一统天下之前，我们需要为同样的字体提供多种格式以兼容不同的浏览器。和竞争激烈的视频格式一样，必须等到某种字体格式明显胜出的时候，才能删除对其他方案的支持。

喜的是，现在给每种浏览器设置对应的自定义字体都很简单。我们来试试！

5.6.2　使用@font-face 嵌入网页字体

我们将用`@font-face` 规则改进一下 And the winner isn't...网站版面。

首先我们得有字体。网上有很多很棒的字体资源站点，有免费的也有收费的。虽然 Google 也提供了免费的网页字体，且基本上都可使用`@font-face` 规则（www.google.com/webfonts），但我个人最喜欢 Font Squirrel（www.fontsquirrel.com）。另外还有 Typekit（www.typekit.com）和 Font Deck（www.fontdeck.com）上也有一些非常优秀的付费字体。



巧合的是，我在网站中使用的来自 Font Squirrel 的字体都是免费的（我承认我抠门 :)）。这些字体包括 Bebas Neue、Bitstream Vera Sans 和 Collaborate Thin。从 Font Squirrel 上下载的`@font-face` 包是一个 ZIP 文件，里面包含该字体各种格式的文件（WOFF、TTF、EOT 和 SVG），以及一个用来演示字体调用规则的 `stylesheet.css` 文件。例如，使用 Bebas Neue 字体的规则如下：

```
@font-face {
    font-family: 'BebasNeueRegular';
    src: url('BebasNeue-webfont.eot');
    src: url('BebasNeue-webfont.eot?#iefix') format('embedded-
opentype'),
         url('BebasNeue-webfont.woff') format('woff'),
         url('BebasNeue-webfont.ttf') format('truetype'),
         url('BebasNeue-webfont.svg#BebasNeueRegular') format('svg');
    font-weight: normal;
    font-style: normal;
}
```

和浏览器私有前缀的原理类似，浏览器会根据自身特性应用列表中能识别的样式，忽略无法识别的样式。用这种方法能保证无论什么浏览器都有一个可用字体。

上面这段代码对"复制粘贴党"来说很方便，但需要注意字体文件的存放路径。比如我一般会将 ZIP 包中的字体文件存放在一个专门的 `fonts` 文件夹，该文件夹与 `css` 文件夹平级。因此我将上面这段代码拷贝到样式表文件中之后，还需要对文件路径做点修改，具体如下所示：

```
@font-face {
    font-family: 'BebasNeueRegular';
    src: url('../fonts/BebasNeue-webfont.eot');
    src: url('../fonts/BebasNeue-webfont.eot?#iefix')
format('embedded-opentype'),
         url('../fonts/BebasNeue-webfont.woff') format('woff'),
         url('../fonts/BebasNeue-webfont.ttf') format('truetype'),
         url('../fonts/BebasNeue-webfont.svg#BebasNeueRegular')
format('svg');
    font-weight: normal;
    font-style: normal;
}
```

接下来就是给相关样式设置正确的字体和粗细（如果需要）。此处我想将导航链接文字的字体修改为 Bebas Neue：

```
nav ul li a {
  height: 42px;
  line-height: 42px;
  text-decoration: none;
  text-transform: uppercase;
  font-family: 'BebasNeueRegular';
  font-size: 1.875em; /*30 ÷ 16 */
  color: black;
}
```

导航栏在浏览器中变成了如下效果：



替换字体之后一般还需要修改字体大小。不过我们之前已经将字体换算过程写在了注释里，那就很容易依此修改了。如果设计图和 CSS 代码都使用同一款字体，那会有一个额外的好处，你可以直接从设计图中获取字体大小。比如我们例子中的设计图对“EVERY YEAR…”开头的这段文字大小设定为 102 像素，所以直接使用百试不爽的“**目标元素尺寸÷上下文元素尺寸=百分比尺寸**”公式，将文字大小转换为相对尺寸：

```
#content h1 {
  font-family: Arial, Helvetica, Verdana, sans-serif;
  text-transform: uppercase;
  font-family: 'BebasNeueRegular';
  font-size: 6.375em;  /* 102 ÷ 16 */
}
```

将所有相关样式中的 `font-family` 和 `font-size` 声明全部修改之后，（使用 WOFF 字体格式的）网页在 Google Chrome 浏览器中的效果如下图所示：



网站的整体效果尚不完美，但现在字体完美还原了设计图中的效果。为了对比效果，我们看看 iPad 2（iOS 4.2 之后的版本支持 TTF 字体格式）上的显示效果：



5.7　帮帮我，标题模糊怎么办

我第一次使用`@font-face` 字体设置网页版面时，这个问题差点没把我逼疯。并非只有响应式设计才会发生标题模糊，而是所有使用`@font-face` 字体的标题都不能幸免。下图是当时网站设计图的部分截图：



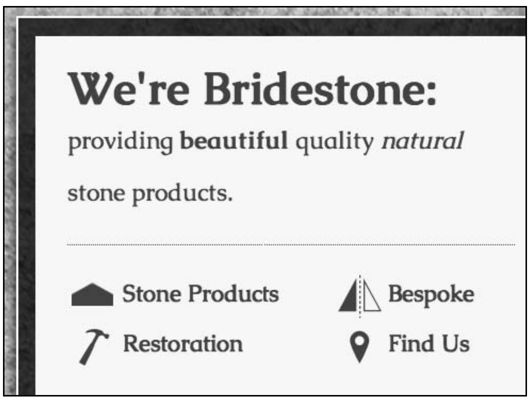
网站制作完成后，相关部分的标签代码如下：

```
<div class="intro">
  <h1>We're Bridestone: <span>providing <b>beautiful</b> quality
      <i>natural</i> stone products.</span></h1>
  …more code…
</div> <!-- intro:END -->
```

相关的 CSS 代码如下：

```
.intro h1 {
  font-family: CaudexBold, "Times New Roman", Times, serif;
  font-size: 2.63636364em;
  line-height: 1em;
}
.intro h1 span {
  font-size: 0.545454545em;
  font-family: CaudexRegular, "Times New Roman", Times, serif;
  font-weight: normal;
}
```

虽然我使用`@font-face`设定了和设计图中完全一样的字体，但浏览器中标题文字看起来还是有一点模糊：



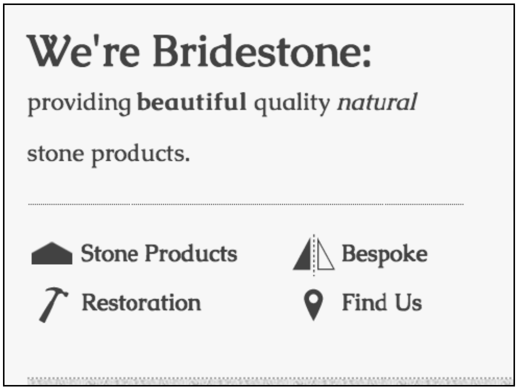
希望你能看出来 We’re Bridestone 这几个词和设计图的效果不太一样。此处的文字更粗，因而清晰度下降了。

后来，我发现导致该问题的原因是字体粗细。除非显式声明`font-weight`属性，否则大多数浏览器都会为标题元素应用标准的`font-weight`（一般都是 700）。因此解决方法就是始终为应用了`@font-face`字体的标题元素设定`font-weight`属性。例如在本例中，我将 CSS 代码修改如下：

```
.productIntro h1 {
  font-family: CaudexBold, "Times New Roman", Times, serif;
  font-weight: 400;
```

```
  font-size: 2.63636364em;
  line-height: 1em;
}
```

这样就会覆盖浏览器默认为标题元素设定的 `font-weight` 值，修改后的效果如下图所示，与设计图效果完全一致：



在响应式设计中使用自定义@font-face 字体的注意事项

使用`@font-face` 自定义网页字体的方法总的来说不错。唯一需要注意的，是在响应式设计中使用该技术时要考虑到字体文件大小。例如，And the winner isn't...网站使用了三种自定义字体：Bebas Neue、Bitstream Vera Sans 和 Collaborate Thin。最坏的情况下，如果设备渲染页面时需要 SVG 字体格式，那相对于使用标准的网页安全字体如 Arial，会增加 70KB 的额外数据。这几种字体其实相当轻量级——有些字体可能会非常庞大！如果你想保持网站的高性能，那请注意控制自定义字体的文件尺寸。

5



真正的响应式字体单位还不能广泛使用

当前的 CSS3 字体模块工作草案中引入了视口相对字体（http://www.w3.org/ TR/ css3-values/#viewport-relative-lengths）。相对单位 vw（视口宽度）、vh（视口高度）和 vm（视口最小边长，即 vm 和 vh 中较小的一个）在获得浏览器广泛支持之后，能为我们节省大量开发时间。悲哀的是目前除了 IE 9 之外，其他浏览器都不支持视口相对单位。

5.8 新的 CSS3 颜色格式和透明度

截止目前，CSS3 已经给予了我们选择元素和添加自定义字体的强大支持。现在我们来看看 CSS3 的颜色使用方法，这些方法在以前简直不可能实现。

首先，CSS3 允许我们使用新方法如 RGB 或 HSL 来声明颜色。另外，我们还能在这两个方法后边追加一个透明通道（分别是 RGBA 和 HSLA）。

5.8.1　RGB 颜色

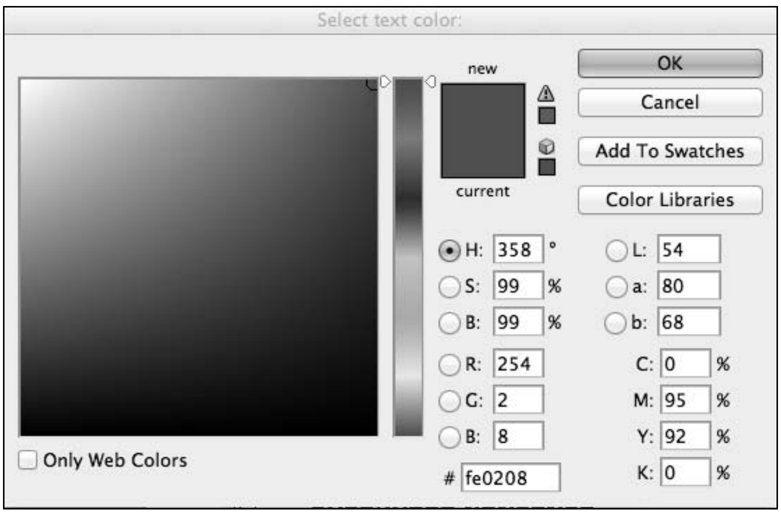
RGB（红绿蓝）是一种已存在了数十年的颜色体系。它的原理是通过定义不同的红绿蓝色值来组成一个颜色。例如，And the winner isn't...网站中奇数导航链接文字的颜色，在 CSS 中被定义为一个十六进制值 `#fe0208`：

```
nav ul li:nth-child(odd) a {
  color: #fe0208;
}
```

在 CSS3 中，该值可以使用 RGB 值来描述：

```
nav ul li:nth-child(odd) a {
  color: rgb(254, 2, 8);
}
```

大多数图片编辑器在它们的颜色选择器中会同时显示颜色的十六进制值和 RGB 值。下面的截图展示了 Photoshop 的颜色选择器，图中的 R、G 和 B 输入框中显示了每个颜色通道的值：



可以看到 R 的值是 254，G 的值是 2，而 B 的值是 8。这个值很容易转换为 CSS3 颜色值：在 CSS 中，颜色模式（如 rgb）定义之后，圆括号中的颜色值必须按照红、绿、蓝这个顺序排列，之间使用逗号分隔。

5.8.2 HSL 颜色

除了 RGB，CSS3 还可使用 HSL（色相、饱和度、亮度）模式来声明颜色。

> **HSL 不是 HSB**
>
> 不要错误地认为图片编辑软件如 Photoshop 中显示的 HSB（色相、饱和度、明度）值就是 HSL——它们不一样[①]！

HSL 被广泛使用是因为它非常容易理解，根据该模式提供的颜色值就能描绘出具体颜色。比如，除非你是某种颜色选择器狂人，否则我敢打赌你肯定无法立即告诉我 `rgb(255, 51, 204)`是什么颜色？有人愿打赌吗？肯定没有，我也不会。但是给我 HSL 的颜色值 `hsl(315, 100%, 60%)`，我可以大概猜出它是位于洋红色和正红色之间的某种颜色（实际上它是一种喜庆的粉红色——我可能开始有点喜欢《红磨坊》了）。我怎么知道？其实很简单……

HSL 模式基于一个 360° 的色相环，第一个数字代表色相，60° 时为黄色，120° 时为绿色，180° 时为青色，240° 时为蓝色，300° 时为洋红色，360° 时为红色。所以前面提到的 HSL 颜色色相为 315，所以很容易看出它介于洋红（300°）和红（360°）之间。其后的两个值分别表示饱和度和亮度，值为百分比，用于改变基础的色相。如果想要更加饱满的颜色，则第二个值使用一个高一点的百分比即可。最后一个值控制亮度，可在 0% 的全黑到 100%的全白之间变化。

因此，定义了 HSL 颜色值之后，在它的基础上创建类似颜色就很方便，只需要修改饱和度和亮度的百分比值即可。例如，我们的红色导航链接可以使用 HSL 颜色值来定义：

```
nav ul li:nth-child(odd) a {
  color: hsl(359, 99%, 50%);
}
```

如果我们想在鼠标悬停时让文字颜色稍微变深一点，就可以使用相同的 HSL 值，然后只修改一下亮度百分比（最后一个值），如下代码片段所示：

```
nav ul li:nth-child(odd) a:hover {
  color: hsl(359, 99%, 40%);
}
```

总之，如果你能记住针对色相环的这个顺口溜：Young Guys Can Be Messy Rascals（或者其他你容易记住的顺口溜[②]），那你基本就能自己写出 HSL 颜色值，或者基于某种颜色创建类似颜色，不用再依赖颜色选择器。在公司聚会时把这个技巧给那些后端 PHP 专家炫

① 具体区别请见: http://tinyurl.com/9qgz8ow。——译者注
② 比如中国人常说的“赤橙黄绿青蓝紫”。——编者注

耀一下，他们会对你肃然起敬。

5.8.3　针对 IE6、IE7 和 IE8 提供备用颜色值

你可能都猜到了，版本 9 以下的 IE 浏览器不支持 RGB 和 HSL。因此，如果需要针对这些浏览器提供备用的颜色声明，则要将其放在 RGB 或 HSL 值之前。例如，为前面提到的导航链接增加备用十六进制颜色值的代码如下：

```
nav ul li:nth-child(odd) a {
  color: #fe0208;
  color: hsl(359, 99%, 50%);
}
```

5.8.4　透明通道

为什么我们不继续沿用已被使用多年的可靠的十六进制颜色值，非要自找麻烦地使用 HSL 或 RGB？有这样的疑惑很正常。HSL 和 RGB 与十六进制颜色值最大的区别，是它们支持透明通道。这意味着可以让元素透明，使其下方的元素可见。

我们来对 And the winner isn't...网站做点修改以说明透明效果。首先，我们给 `body` 元素设置一个很烂的背景图，如下：

```
body {
  background: url(../img/grunge.jpg) repeat;
}
```

然后，我们给`#wrapper` div（它包裹所有其他页面元素）添加一个白色的背景。不过此处我们不会使用十六进制的白色，而是使用 HSLA 颜色值，具体如下面代码片段中加粗的代码行所示：

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 96%; /* 最外层的DIV */
  max-width: 1414px;
  background-color: hsla(0, 0%, 100%, 0.8);
}
```

HSLA 颜色声明与标准的 HSL 规则类似。不过颜色必须得声明为`hsla`模式（而不是`hsl`），增加一个额外的透明度值，该值的格式是一个介于 0（全透明）到 1（不透明）之间的小数。上面代码中我们将白色的`#wrapper` 设置为半透明。浏览器中的显示效果如下所示：



RGBA 的语法和 HSLA 的基本一样，即在颜色值后追加一个透明度值：

```
background-color: rgba(255, 255, 255, 0.8);
```

希望你已经看出，为 RGB 和 HSL 颜色模式增加透明通道，能让我们快捷灵活地处理分层叠加元素。也就是说我们不用再依赖透明图片（如 PNG 或 GIF 图片）来实现这类视觉效果，这对制作响应式设计是个绝好的消息。

为什么不使用 opacity

CSS3 还允许通过 `opacity` 声明来设置元素的透明度。该透明度的值也是一个介于 0 到 1 之间的小数（如将 `opacity` 设置为 0.1 表示为 10%透明）。但是这种透明度与 RGBA 及 HSLA 有所不同，这种方式设置的透明度会对整个元素产生影响（元素的内容都会透明）。反之，使用 HSLA 或 RGBA 则可以仅让元素的某些部分有透明效果。这样，一个元素可以带有 HSLA 透明背景，但内部的文字仍然不透明。

CSS3 颜色模块是第一个成为推荐标准的 CSS3 模块。因此和 CSS3 选择器模块一样，CSS3 颜色模块现在就可以放心使用，该模块的实现方法之后不会再有什么变化。

5.9　小结

在本章，我们学习了如何使用新的 CSS3 选择器来选择页面中任何我们想要的元素。我们还学习了如何在超短时间内制作可响应的多栏内容布局，以及如何解决一些常见的烦人问题（如超长 URL 换行之类）。此外我们还研究了新的 CSS3 颜色模式，以及如何应用带有透明通道的 RGB 和 HSL 颜色以制作优雅的视觉效果。本章我们还学习了如何使用 @font-face 规则来为网页添加自定义字体，终于将我们从多年来不得不使用网页安全字体的桎梏中解放了出来。尽管学习了这么多高级的新特性和新技巧，其实我们仅了解了 CSS3 的一点皮毛。接下来我们将继续学习如何使用 CSS3 的文字阴影、盒阴影、渐变以及多重背景图片，让我们的响应式设计更快速、高效、容易维护。

用 CSS3 创造令人惊艳的美 6

上一章，我们学习了一些简捷有效的 CSS3 技巧，对制作响应式设计很有帮助。另外还使用 CSS3 的`@font-face` 规则为网页添加了自定义字体，使其视觉效果大大提升。此外还学习了 CSS3 中选择 DOM 元素的工具。在学习了这些基础知识之后，我们来看看 CSS3 的一些高级特性，这些特性能给我们的响应式设计带来显著的美学提升，而且大部分都不需要图片，能为我们节省宽带。

本章内容

- 使用 CSS3 制作文字阴影
- 使用 CSS3 制作盒阴影（即元素投影）
- 使用 CSS3 制作渐变背景
- 使用 CSS3 的多重背景图片
- 使用 CSS3 背景渐变来制作图案
- 使用 CSS3 的`@font-face` 规则来制作节省带宽的图标

在此，我要重申一遍为什么 CSS3 对响应式设计非常有用：使用 CSS3 替代图片，在有带宽限制的网页中可有效减少 http 请求（从而使网页加载更快），并可使网页更灵活、更容易维护。这些优点甚至在传统的固定宽度桌面版网页中也很有用，但它们在响应式设计中更为重要，因此它们可以让网页在不同的视口中显示不同尺寸的模块或文字阴影，而不需要为此单独制作和导出图片。相信你肯定也这么想，那就开始吧。

> **浏览器私有前缀**
>
> 在开发 CSS3 时，要记住添加相关的浏览器私有前缀以保证最广泛的浏览器兼容。除了这种方法，如果你愿意在代码中插入一点 JavaScript，则可以考虑之前说过的-prefix-free 脚本。该脚本会自动为 CSS3 规则追加浏览器私有前缀，这样你在样式表中就可以只写 W3C 规定的标准代码。该脚本地址为：http://leaverou.github.com/prefixfree/。

6.1　文字阴影

文字阴影是最被广泛支持的 CSS3 特性之一。和`@font-face`一样，它有过一段短暂的前生，之后在 CSS 2.1 中被废弃。好在，它再次投胎转世，并获得了广泛支持（所有现在浏览器及 Internet Explorer 9 之后的 IE 均支持它）。

我们来看看文字阴影的基本语法：

```
.element {
    text-shadow: 1px 1px 1px #cccccc;
}
```

记住，阴影值的速记规则永远是先向右再向下。因此，第一个值指的是右侧阴影的大小，第二个值指的是下方阴影的大小，第三个值指的是模糊距离（即阴影从开始变淡到完全消失的距离），最后一个值是阴影颜色。

6.1.1　HEX、HSL 或 RGB 颜色都可以

阴影颜色不一定非得是十六进制值，也可以是 HSL(A)或 RGB(A)：

```
text-shadow: 4px 4px 0px hsla(140, 3%, 26%, 0.4);
```

但要注意浏览器必须得同时支持 HSL/RGB 颜色和`text-shadow`才能渲染出效果。考虑到浏览器兼容性，在使用 HSLA 或 RGBA 阴影时我一般都这样做：

```
text-shadow: 4px 4px 0px #404442;
text-shadow: 4px 4px 0px hsla(140, 3%, 26%, 0.4);
```

即先定义一个使用十六进制颜色的阴影（作为针对老版本浏览器的备用方案），然后再定义一个使用 HSLA 或 RGBA 颜色的阴影。

6.1.2　px、em 或 rem 都行

阴影值也可以使用`em`或`rem`单位。如下所示的 **AND THE WINNER ISN'T** 网站的设计图：



在 Photoshop 中，EVERY YEAR 这几个字的大小是 102 像素，文字阴影大小是 4 像素。因此使用我们百试不爽的**目标元素尺寸÷上下文元素尺寸=百分比尺寸**这个公式来计算一下（4÷102=.039215686），结果如下：

```
text-shadow: .039215686em .039215686em 0em #dad7d7; /* 4 ÷ 102 */
```

下图展示了浏览器中的效果：



6

我个人其实很少用 em 或 rem 作为阴影大小的单位。因为阴影一般都比较小，一个或两个像素的阴影，在所有视口中的效果都很不错。

6.1.3　取消文字阴影

如果你眼神够好，就会发现右侧内容区的第二行文字，WHEN I WATCH THE OSCARS I'M ANNOYED…. ，也应用了文字阴影。原因如下：

```
<h1>Every year <em>when I watch the Oscars I'm annoyed...</em></h1>
```

当前的文字阴影效果应用在整个<h1>标签上（该标签内包含<em>标签），所以我们需要取消<em>标签上的 text-shadow：

```
#content h1 em {
    font-family: 'BitstreamVeraSansRoman';
    display: block;
    line-height: 1.052631579em; /* 40 ÷ 38 */
    color: #757474;
    font-size: .352941176em; /* 36 ÷ 102 */
    text-shadow: none;
}
```

修改后的效果如下图所示：



左上方阴影

使用负值可以制作出左上方阴影效果，例如：

```
text-shadow: -4px -4px 0px #dad7d7;
```

效果如下图：



如果不需要阴影模糊效果，可以将 `text-shadow` 的第三个值（模糊半径）从声明中删除，如下所示：

```
text-shadow: -4px -4px #dad7d7;
```

这种简写假定第三个值没有声明，而前两个值表示阴影偏移距离。

6.1.4 制作浮雕文字阴影效果

我一直觉得 `text-shadow` 最适合用来制作浮雕文字。这种效果一般最适合应用在非白色背景的深色文字上，搭配以高亮颜色（如纯白色或类似颜色）的阴影。我们来给网站的导航链接添加浮雕效果试试：

```
nav ul li a {
    height: 42px;
    line-height: 42px;
    text-decoration: none;
    text-transform: uppercase;
    font-family: 'BebasNeueRegular';
    font-size: 1.875em; /*30 ÷ 16 */
    color: #000000;
    text-shadow: 0 1px 0 hsla(0, 0%, 100%, 0.75);
}
```

效果如下图所示，可以说是恰到好处——有了一点凹陷效果，同时又没有过分炫耀文字阴影！



6

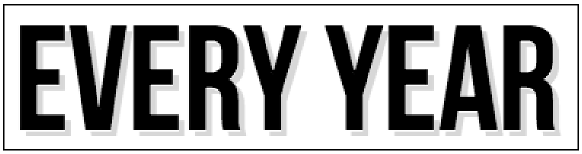
> 想要最好的浮雕文字效果，我有诀窍：不要模糊，不要水平阴影，仅在垂直方向设置 1 或 2 像素的“白影”即可。

6.1.5　多重文字阴影

我们还可以制作多重文字阴影，只需将两组值使用逗号分隔开即可，比如：

```
text-shadow: 0px 1px #ffffff,4px 4px 0px #dad7d7;
```

之前精妙的浮雕效果必不可少，否则文字会变模糊。所以这个声明合并了已有的投影效果和上一节的浮雕效果。浏览器中的效果如下图所示：



> 想要了解 W3C 对 `text-shadow` 属性的标准定义，请参阅 http://www.w3.org/TR/css3-text/#text-shadow。

6.2　盒阴影

掌握了文字阴影，盒阴影就是小菜一碟了。盒阴影的语法与文字阴影完全一样：水平偏移距离、垂直偏移距离、模糊半径，以及阴影颜色：

```
box-shadow: 0px 3px 5px #444444;
```

但是，盒阴影的的跨浏览器支持并不好，所以明智的做法是使用浏览器私有前缀，例如：

```
-ms-box-shadow: 0px 3px 5px #444444;
-moz-box-shadow: 0px 3px 5px #444444;
-webkit-box-shadow: 0px 3px 5px #444444;
box-shadow: 0px 3px 5px #444444;
```

我们来给 AND THE WINNER ISN’T 网站侧边栏的电影海报加点阴影效果：

```
.sideBlock img {
    max-width: 45%;
    box-shadow: 0px 3px 5px #444444;
}
```

浏览器中的效果如下：



6.2.1 内阴影

`box-shadow` 属性可以用来制作内阴影——出现在元素内部，而不是外部。内阴影可用来制作光晕效果，如下所示：

```
box-shadow:inset 0 0 40px #000000;
```

语法和之前的盒阴影差不多，只是多出来的 `inset` 告诉浏览器设置内阴影效果。现在给 <body>标签应用该规则，为整个页面添加光晕效果，这样就会让页面的四周都出现阴影。

```
body {
    -moz-box-shadow:inset 0 0 30px #000000;
    -webkit-box-shadow:inset 0 0 30px #000000;
    box-shadow:inset 0 0 30px #000000;
}
```

浏览器中的效果如下：



6.2.2　多重阴影

和文字阴影一样，盒阴影也可以有多重阴影效果。语法也类似，即将两组值用逗号分开，这样两组阴影就会按照代码中的先后顺序从上到下应用到元素上。换句话说，就是代码中先声明的规则，在浏览器中会覆盖下面的规则。

```
box-shadow: inset 0 0 30px hsl(0, 0%, 0%),
            inset 0 0 70px hsla(0, 97%, 53%, 1);
```

我将这句代码添加到 body 上，就能制造出一种神秘的粉红闺房效果。这肯定是受了页面上的《红磨坊》海报图片的影响！



6

可以说，本节的讲解简单明了。这也恰恰说明了 CSS3 在实现设计创意方面的强大能力。给元素追加或删除视觉效果就是一两秒钟的事，根本不需要使用图像处理软件。

> 想要了解 W3C 对 box-shadow 属性的标准定义，请参阅 http://www.w3.org/TR/css3-background/#the-box-shadow。

6.3　背景渐变

在没有 CSS3 的时候，如果想做一个背景渐变效果，就得使用一个很细的渐变切片进行水平/垂直平铺。对于使用图片而言，这确实是一种很好的折中方案。一张仅有一两像素宽的图片，不会给宽带造成太大负担，而且它可以用在网站的多个元素上。

6.3.1　线性背景渐变

使用上面说过的技巧，来给 AND THE WINNER ISN'T 网站的侧边栏制作一个背景渐变效果：

```
aside {
    border-right-color: #e8e8e8;
    border-right-style: solid;
    border-right-width: 2px;
    margin-top: 58px;
    padding-left: 1.5%;
    padding-right: 1.0416667%;
    margin-left: 1.0416667%;
    float: left;
    width: 20.7083333%;
    background: url(../img/sidebarBg2.png) 50% repeat-x;
}
```

浏览器中的效果如下：



如果想修改渐变效果，还得使用图像编辑软件。另外，内容可能因为超出固定尺寸限制而偶尔溢出渐变背景。这个问题在响应式设计中简直无法忍受，因为我们想让页面结构能随意改变形状（比如变高或变宽），但不会破坏设计效果。

例如，假设我想给侧边栏中的每个模块再追加两部电影。效果如下：



效果不是太差，但灰色渐变没有按照我喜欢的那样覆盖整个侧边栏。这时我又不得不回到图像编辑软件中重新制作渐变图片。如果使用 CSS3 渐变，就会灵活很多。用纯 CSS3 就可以做出和上面一样的渐变，而且不用图片，代码如下：

```
background: linear-gradient(90deg, #ffffff 0%, #e4e4e4 50%, #ffffff 100%);
```

在支持该特性的浏览器中，效果如下所示：



不论这块区域有多高（毕竟，评分差不多的好电影和烂电影有很多），CSS3 渐变始终能覆盖整个区域。

背景渐变唯一美中不足的是它不像其他一些 CSS3 特性那样被广泛支持。比如 IE 9 就没有对它的原生支持（微软承诺在 IE 10 中支持）。其他大多数浏览器都支持背景渐变，不过都是以私有前缀方式。不过这并不足以阻止你使用该特性来增强设计效果，因为有很多浏览器现在已经支持该特性，而剩下的浏览器在不久的将来也会支持。作为老版本浏览器的降级方案，最好先定义一个固定背景颜色，这样老版本浏览器在无法识别渐变规则的时候，至少会渲染出一个实色的背景。

注意：过去曾有几种不同的背景渐变语法

过去，针对同样的背景渐变效果，不同的浏览器厂商使用不同的语法来实现。Webkit 是这个问题的罪魁祸首，不过庆幸的是，从 Safari 5.1 开始，Webkit 采用了和 Mozilla 一样的做法——也就是 W3C 公布的做法。

分解线性渐变语法

线性背景渐变语法可能有点复杂，所以我们对其做一个分解：

- 圆括号中的第一个值（即例子中的 `90deg`，可选）定义了渐变的方向。不定义该值则默认是一个垂直从顶部到底部的渐变。你还可以使用如 `to top right` 这样的值，这会产生一个朝向右上角的对角线渐变。
- 下一个值（例子中的`#ffffff 0%`）定义的是渐变的“起点”，包括起点的颜色和位置。你也可以使用像 `blue 20%`这样的值，这样就是从蓝色开始渐变到下一个颜色，而渐变开始位置则位于假想的渐变路径的 20%处。同样，起点位置也可以使用负值，这样渐变从实际可见区域之外就开始了。比如：

```
background: linear-gradient(90deg, #ffffff -50%, #e4e4e4 50%, #ffffff 100%);
```

  上面这行代码意味着渐变从可见区域之外开始，假想的渐变路径比可见区域长 50%。

- 下一个值指的是“过渡颜色点”。我们来回顾一下：沿着 90 度垂直方向，从白色开始（`(#ffffff 0%`），向位于渐变路径 50%处的`#e4e4e4` 这个颜色（浅灰色）渐变。这里就是渐变中的第一个过渡颜色点。如果需要的话，可以在渐变“终点”之前定义更多的过渡颜色点（使用逗号分隔）。
- 圆括号中的最后一个值始终是渐变的“终点”。不论在起点之后放置了多少个过渡颜色点，最后一个值始终是终点。

想要了解线性渐变的 W3C 规范，请参阅 http://dev.w3.org/csswg/css3-images/#linear-gradients。

6.3.2 径向背景渐变

CSS3 背景渐变不只局限于线性渐变，制作径向渐变同样简单。径向渐变是从一个中心点开始，依据椭圆形或圆形进行扩张渐变。

径向背景渐变的语法如下所示：

```
background: radial-gradient(center, ellipse cover, #ffffff 72%, #dddddd 100%);
```

将这个声明追加到`#content` 规则中，产生的效果如下图所示：



看到页面四角慢慢变暗的效果了吧？这就是径向渐变。我们来对其做一个语法分解，看看它是如何运作的。

6

分解径向渐变语法

在 background 属性之后，我们设定的值是 `radial-gradient`（而不是 `linear-gradient`）。然后在圆括号中设定起点。在上一节的例子中我们用的是 `center`，其实也可以使用如 `25px 25px` 这样的值，这就表示从距元素上边和左边均为 25 像素的那个点开始渐变。例如：

```
background: radial-gradient(25px 25px, ellipse cover, #ffffff 72%, #dddddd 100%);
```

这行代码产生的效果如下图所示：



中心点距离元素上边和左边均为 25 像素，然后向四周平滑辐射。

声明中的下一个值更简单，它指的是径向渐变的形状和大小：

```
background: radial-gradient(center, ellipse cover, #ffffff 72%, #dddddd 100%);
```

渐变形状要么是 `circle`（圆形，渐变会均匀地向各个方向辐射），要么是 `ellipse`（椭圆形，在不同的方向辐射量不同）。而渐变形状的大小则有很大的灵活性，大小值可以是下列任何一种。

- `closest-side`：（渐变形状是圆形时）以距离中心点最近的一边为渐变半径，或者（渐变形状是椭圆形时）以距离中心点最近的水平或垂直边为渐变半径。
- `closest-corner`：以距离中心点最近的一角为渐变半径。
- `farthest-side`：和 `closest-side` 正好相反，（渐变形状是圆形时）以距离中心点最远的一边为渐变半径，或者（渐变形状是椭圆形时）以距离中心点最远的水平或垂直边为渐变半径。
- `farthest-corner`：以距离中心点最远的一角为渐变半径。
- `cover`：和 `farthest-corner` 完全一样。
- `contain`：和 `closest-side` 完全一样。

接下来是定义渐变起点、过渡颜色点以及终点（这部分和线性渐变是一样的）。

例如，我们将之前的规则改成如下这样：

```
background: radial-gradient(20px 20px, circle cover,
                            hsla(9,69%,85%,0.5) 0%,
                            hsla(9,76%,63%,1) 50%,
                            hsla(10,98%,46%,1) 51%,
                            hsla(24,100%,50%,1) 75%,
                            hsla(10,100%,39%,1) 100%);
```

可以看到，我们是以距离左边和上边均为 20 像素的点为中心点，使用圆形渐变，渐变半径覆盖整个元素显示区域，并使用了多组 HSL(A)的过渡颜色点。效果如下：



这效果从美学角度讲确实不敢恭维,但我希望这能说明使用纯 CSS3 制作视觉效果的无穷威力。

> 想要了解 W3C 对背景渐变的标准定义，请参阅 http://dev.w3.org/csswg/css3-images/#radial-gradients。

> **制作完美 CSS3 渐变的简便方法**
>
> 如果写出一个 CSS3 渐变对你来说有点难度，那你可以考虑使用在线渐变生成器。我个人最喜欢的渐变生成器是 http://www.colorzilla.com/gradient-editor/。它的用户界面风格很像图像处理软件，你可以选择颜色、过渡点、渐变风格（线性渐变或径向渐变），甚至可以选择自己喜欢的颜色空间（HEX、RGB(A)、HSL(A)）。这个在线渐变生成器还会加载一些预置的渐变供你选用和修改。如果这些还不足以打动你，须知它还提供了可选的代码设置，用来修正 IE 9，让它显示渐变效果，而且还为老版本浏览器提供了备用的背景颜色。还是不动心？它还能基于已有的图片直接生成渐变，这个功能怎么样？我想这下总该打动你了。

6.3.3 重复渐变

CSS3 还可以让我们制作重复背景渐变效果。我们来看看如何实现：

```
background: repeating-linear-gradient(90deg, #ffffff 0px, hsla(0, 1%, 50%,0.1) 5px);
```

在侧边栏上应用后的效果如下：



首先给 `linear-gradient` 或 `radial-gradient` 前面加一个前缀“repeating”，后面的语法和普通渐变一样。此处我在白色和灰色过渡颜色点之间使用了像素距离（分别是 `0px` 和 `5px`），百分比也可以。为了达到最好效果，建议在同一个渐变中使用相同的度量单位（像素或百分比）。

我们来试着写一个重复径向渐变：

```
background: repeating-radial-gradient(2px 2px, ellipse,
            hsla(0,0%,100%,1) 2px, hsla(0,0%,95%,1) 10px,
            hsla(0,0%,93%,1) 15px, hsla(0,0%,100%,1) 20px);
```

语法和之前标准的径向渐变相似。我只是修改了渐变起点，删除了此处不需要的“cover”值，然后给每个过渡颜色点设置了像素距离。例子中的终点距离是 20 像素，所以渐变每 20 像素就会重复一次。应用在 `body` 上的效果如下图所示。我先警告你——效果不好看！



想要了解 W3C 对重复渐变的标准定义，请参阅 http://dev.w3.org/csswg/css3-images/#repeating-gradients。

另外，还有一种使用背景渐变的方法，我很想分享给你。

6.4 背景渐变图案

我发现自己经常使用精妙的线性渐变，而很少使用径向渐变和重复渐变。当然，到底用什么渐变取决于设计需要。一些聪明的家伙已经将这些背景渐变技巧收集起来整理成了背景渐变模式。我们来看一个例子，将刚才给 body 设置的重复渐变替换为下面的样式：

```
body {
    background-color:white;
```

```
    background-image:
      radial-gradient(hsla(0, 0%, 87%, 0.31) 9px, transparent 10px),
      repeating-radial-gradient(hsla(0, 0%, 87%, 0.31) 0,
                hsla(0, 0%, 87%, 0.31) 4px, transparent 5px,
                transparent 20px, hsla(0, 0%, 87%, 0.31) 21px,
                hsla(0, 0%, 87%, 0.31) 25px, transparent 26px,
                transparent 50px);
    background-size: 30px 30px, 90px 90px;
    background-position: 0 0;
}
```

浏览器中的效果如下图：



看起来不错吧？仅用几行代码，我们就用刚才学习的背景渐变技巧做出了一个可修改的、灵活的背景图案。

CSS 高手 Lea Verou 收集了一系列 CSS3 背景渐变图案，具体请见 http://lea.verou.me/css3patterns/。



6.5 CSS3 的响应性

请大家务必记住，我们可以针对不同的视口使用不同的声明。例如，针对较小的视口可以继续使用上一节中的渐变图案：



但针对大视口（例如宽度大于等于 768 像素的视口），可以不继续使用上图中的渐变图案。因此可以使用媒体查询为大视口设定一个具体规则：

```
@media screen and (max-width: 768px) {
  body {
    background-color:white;
    background-image:
      radial-gradient(hsla(0, 0%, 87%, 0.31) 9px, transparent 10px),
      repeating-radial-gradient(hsla(0, 0%, 87%, 0.31) 0,
      hsla(0, 0%, 87%, 0.31) 4px, transparent 5px, transparent 20px,
```

```
      hsla(0, 0%, 87%, 0.31) 21px, hsla(0, 0%, 87%, 0.31) 25px,
      transparent 26px, transparent 50px);
    background-size: 30px 30px, 90px 90px;
    background-position: 0 0;
  }
}
```

请时刻牢记，只要你有需要，媒体查询就能让你给任何元素针对各种不同视口设定具体规则。这样做都是为了给用户提供最佳的使用体验。

> **使用 CSS 预处理器轻松编写 CSS3 代码**
>
> CSS3 规则目前都需要各种不同的浏览器私有前缀。除了将各种声明的不同前缀写法保存为代码片段，或者使用 JavaScript 文件来自动追加前缀之外，还有一种快速编写 CSS3 代码的方法，就是使用 CSS 预处理器，如最流行的 SASS 和 LESS。比如使用支持 SASS 的 Compass 插件，编写一个盒阴影就变得如此简单：`element { @include box-shadow; }`。当最终生成 CSS 代码时，代码中会包含一组完整的浏览器私有规则，并包含针对 Internet Explorer 的 hack（如果需要的话）。如果这还不足以打动你，那再想想这种预处理器能够让你在 CSS 中使用变量和 `if/while` 这样的编程语句。欲了解更多信息，请前往 SASS 的网站 http://sass-lang.com 和 LESS 的网站 http://lesscss.org。

6.6 组合使用 CSS3 属性

之前，我们主要学习了各种 CSS3 特性的一般用法。下面我们将组合这些特性来制作 THESE SHOULD HAVE WON>>这个超链接的效果。在 AND THE WINNER ISN'T 网站的原始设计图中，该链接按钮的文字使用的是自定义字体，这点我们已经在第 5 章处理过。除此之外，该链接按钮还有一个红色的渐变背景，使用了圆角效果并有一点阴影。该超链接目前的样式如下：

```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
}
```

首先，针对老版本浏览器为其添加一个背景颜色。这样就算浏览器无法渲染渐变，它至少也可以有一个红色的背景。此处我会刻意使用十六进制颜色值，因为如果老版本浏览器不识别渐变的话，那它也不可能支持 RGB 和 HSL 颜色模式：

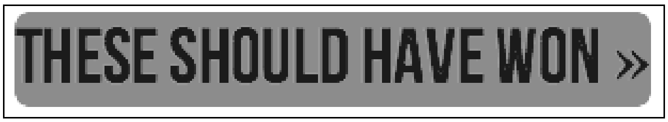
```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
```

```
    background-color: #b01c20;
}
```

接下来添加圆角效果。此处及后续章节中都请注意，我在代码示例中添加的所有 CSS3 属性，一般都需要设定浏览器私有前缀，为简洁起见我在此处省略了前缀：

```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
    background-color: #b01c20;
    border-radius: 8px;
}
```

截至目前我们做出的效果如下：



接下来，将文字颜色改为白色（再强调一次，我想确保在老版本浏览器中也看到同样效果，所以此处使用了简单但兼容的颜色定义），然后设置一定的内边距（也可以使用百分比值）以便让文字四边都有一定的间距：

```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
    background-color: #b01c20;
    border-radius: 8px;
    color: white;
    padding: 30px;
}
```

至此效果如下：

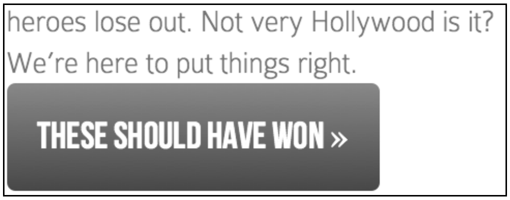


此时，添加内边距导致链接上方的文字被盖住了，所以我们给其追加一个 `float: left`，同时设置背景渐变：

```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
    background-color: #b01c20;
```

```
    border-radius: 8px;
    color: white;
    padding: 30px;
    float: left;
    background: linear-gradient(90deg, #b01c20 0%, #f15c60 100%);
}
```

现在看起来有点模样了：



之后再追加一点上边距，设置一下阴影效果：

```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
    background-color: #b01c20;
    border-radius: 8px;
    color: white;
    padding: 30px;
    float: left;
    background: -moz-linear-gradient(90deg, #b01c20 0%, #f15c60 100%);
    margin-top: 30px;
    box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
}
```

再在浏览器中查看一下，效果已经差不多了：



接下来，我要添加一点文字阴影和一个很细的白边，虽然原始设计文件没有这些效果，但加上之后会带来些许浮雕效果。这就是使用 CSS3 的美妙之处——不用图片，能在工作中轻松快速地做出适当调整。

```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
    background-color: #b01c20;
    border-radius: 8px;
```

```
    color: white;
    padding: 30px;
    float: left;
    background: -moz-linear-gradient(90deg, #b01c20 0%, #f15c60 100%);
    margin-top: 30px;
    box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
    text-shadow: 0px 1px black;
    border: 1px solid #bfbfbf;
}
```

现在我们的按钮在 Firefox 8 中的效果如下：



还有最后一个问题，原始设计图中的那两个尖括号（即 HTML 中的`»`）使用了与按钮文字不一样的字体。但在此处为这么一个特殊符号专门引入一种字体不太划算，所以我准备将这个符号用一个内联标签包裹起来，然后增大它的字号。修改后的标签代码如下：

```
<a href="#">these should have won <span>&raquo;</span></a>
```

用来修正字体大小的对应 CSS 代码如下：

```
#content a span {
    font-size: 1.3em;
}
```

最后完成的效果还是很不错的：



使用 CSS3 替代图片来实现这种效果，最赞的地方在于，可以在超链接中包含任何内容，而它的效果依然保持完美：



6.7　多重背景图片

有一种很常见的设计需求，就是制作一个顶部和底部使用不同背景图片的页面。或者也可能是要求页面内的某个内容模块的顶部和底部使用不同的背景图片。这个需求看起来很简单，使用 CSS 可以轻松地做出这种效果。但是使用 CSS 2.1 来制作这种效果往往需要额外的标签。例如，在使用 CSS3 之前，我解决该问题的一贯方法如下：

```
<body class="headerBackgroundHere">

<div class="footerBackground">
  <div id="container">
    <header>
      // 页眉内容
    </header>
    <div id="main" role="main">
      // 主内容
```

```
    </div>
    <footer>
      // 页脚内容
    </footer>
  </div>

</div> <!--! end of .footerBackground -->

</body>
```

你会看到整个内容区域（也就是 id 为 container 的 div）被包裹在一个类名为 footerBackground 的 div 中。有了这样的代码结构，我们就可以先给 body 设定一个 CSS 规则，用来设置页面顶部的背景图片：

```
body {
    background-image: url("../img/topSlice.png");
    background-repeat: repeat-x;
}
```

然后再给 footerBackground 设定 CSS 规则。此处用来放置页面底部的背景图片：

```
.footerBackground {
    background-image: url("../img/bottomSlice.png");
    background-repeat: repeat-x;
    background-position: bottom;
}
```

这种方法非常好用且基本兼容所有浏览器。但是我一向不赞成仅为解决表现问题而增加额外的标签。

幸好有了 CSS3，我们可以轻松解决这个问题了。因为 CSS3 允许为一个元素设定多重背景（多重背景是 CSS3 背景和边框模块的一部分）。除了 IE 8 及更低版本外，其他浏览器均支持该特性。语法如下：

```
background:
    url('../img/1.png'),
    url('../img/2.png'),
    url('../img/3.png');
```

和多重阴影的排列顺序一样，排在最前面的图片在浏览器中显示时会覆盖在最上面。你还可以在声明中追加背景颜色，像下面这样：

```
background:
    url('../img/1.png'),
    url('../img/2.png'),
    url('../img/3.png') left bottom, black;
```

将颜色定义在最后，这样背景色就会显示在所有背景图片的下面。

不支持多重背景规则的浏览器（如 IE 8 及更低版本）会直接忽略整条规则。所以你可以在 CSS 多重背景规则之前再添加一条普通背景规则来作为针对老版本浏览器的备用方案。

使用多重背景图片时，如果你使用了透明的 PNG 图片，那透过任何叠放在其他图片之上的透明图片都可以看到下面的图片。但是背景图片不一定非要叠放在另一个背景图片之上，图片大小也不用完全一样。

6.7.1　背景图片大小

要设置每个背景图片的大小，使用 `background-size` 属性。使用多重背景时，其语法如下：

```
background-size: 100% 50%, 300px 400px, auto;
```

规则中声明了每张背景图片的大小（先是宽度，再是高度），多组值之间使用逗号分隔，按照背景属性中图片的顺序对应排列。如上面的代码，图片大小值可以使用百分比或像素值，以及如下几个预定义值：

- `auto`：使用图片的原始大小；
- `cover`：按照原始图片的长宽比缩放图片以填充整个元素区域；
- `contain`：按照原始图片的长宽比缩放图片以使其较长的一边适应元素大小。

6.7.2　背景图片位置

另外还可以给不同的背景图片设置不同的位置。我们可以这样做：

```
background:
    url('../img/1.png') center,
    url('../img/2.png'),
    url('../img/3.png') left bottom, black;
```


上例中的第二张图片没有声明背景位置，因此会使用默认位置 `top left`。

6.7.3　背景属性的缩写语法

可以使用缩写语法将各种背景属性组合起来。但是以我的经验看，这样会产生捉摸不定的结果。所以我更倾向于使用普通写法，先声明多重背景图片，然后声明背景大小，最后声明背景位置。

> W3C 有关多重背景的文档，请参阅此处：http://www.w3.org/TR/css3-background/#layering。W3C 有关背景大小的文档，请参阅此处：http://www.w3.org/TR/css3-background/#the-background-size。W3C 有关背景位置的文档，请参阅此处：http://www.w3.org/TR/css3-background/#the-background-position。

6.8　更多 CSS 特性

到现在为止，我们只讲解了一小部分 CSS3 特性。但是，这一部分是在现实世界中最有用处的。同时，我认为这些特性也是在响应式设计中用来高效灵活地制作视觉效果的最有效手段。不过，你应该一如既往地关注除我们所见的这片“小树林”之外的更广阔的 CSS3“森林”，这其中必有一些东西能激起你的兴趣。

6.9　可缩放图标：响应式设计中的完美选择

聪明的人们已经用 CSS3 实现了很多超棒的效果。其中有一个我非常喜欢且现在经常使用的技巧就是`@font-face` 图标。

你可能会疑惑：“这是什么东西？”好奇的朋友，让我来告诉你。还记得我们在上一章中使用CSS3的`@font-face`来给网页应用自定义字体吗？`@font-face`图标就是将字符做成常用图标的特定字体。以往我们都是使用很多单独的图标图片，或者将图标图片组合成一张很大的雪碧图，而`@font-face` 图标则可以让你只用一种字体就涵盖所有要用的图标（这样就只需要一个 http 请求，唔哈哈！）。另外，因为使用的是字体，所以图标就可以完美缩放了——这与响应式设计真可谓天作之合。Fico 上就有很好的`@font-face`图标，详情请见 http://fico.lensco.be/。



6.10 小结

本章讲解了 CSS3 的很多新特性。CSS3 渐变背景让我们可以使用纯代码来制作非常好看的背景效果，甚至可以设计出背景图案。我们还学习了如何使用 `text-shadow` 来制作浮雕文字，还学习了如何使用 `box-shadow` 来给元素的外部和内部添加阴影效果。

在响应式设计中，使用纯 CSS3 来制作这些视觉效果还会带来额外好处，这意味着元素效果不会像以往使用图片那样既费宽带又难维护。当然，有时使用图片是不可避免的，但 CSS3 为我们提供了非常大的灵活性。例如，本章我们使用 CSS3 多重背景图片给页面添加了多重背景，并分别指定了背景图片位置。这一技术结束了必须使用额外标签的历史。另外，我们使用这些特性来给响应式设计添加了一些酷炫的效果，在现代浏览器中，不论视口大小，都可尽情享受这些精妙的效果和美感。但在恼人的老版本浏览器如 Internet Explorer 中，无法渲染这些效果，不过问题倒也不严重。

截至目前，我们对 CSS3 特性的探索都停留在静态效果上：将页面上的元素排列在某个固定位置，并保持一种固定状态。其实，CSS3 还可以做更多事情。下一章，我们将学习把一个元素从一种状态转换到另一种状态的方法，并且探索 CSS 此前从未涉及的领域：动画。

6

CSS3 过渡、变形和动画 7

在前面两章中，我们学习了一些 CSS3 的新特性和新功能。不过到目前为止，我们看到的所有效果都是静态的。其实 CSS3 能做的远不止于此。

目前的情况是，如果页面上需要一些动画效果，要么你自己编写 JavaScript，要么使用 JavaScript 框架（如 jQuery）来提高效率。但是，一些熟悉 CSS3 的同志早就看不惯动不动就用 JavaScript 的做法了，他们正想方设法收复失地。虽然 CSS3 不可能在短期内取代 jQuery 或类似的框架，但它完全有能力做一些如平滑过渡（比如在鼠标悬停时）和在屏幕上移动元素之类的事情。这对我们来说是个好消息，它意味着在越来越多支持现代浏览器的设备中，我们可以使用 CSS 替代 JavaScript 实现动画效果。好吧，我的意思是：你完全可以从“待办事项”中将“学习如何使用 jQuery 制作动画效果”这项划掉，因为用纯 CSS 能完成同样有趣的效果。和之前一样，不支持这些特性的浏览器不会受到任何影响，它们会自动跳过无法识别的规则，就好像这些代码根本不存在一样。

本章内容

- 什么是 CSS3 过渡以及如何使用它
- 如何编写 CSS3 过渡以及它的缩写语法
- CSS3 过渡时间函数（ease、cubic-bezier 等）
- 响应式网站中有趣的过滤效果
- 什么是 CSS3 变换以及如何使用它
- 理解不同的 2D 变换（scale、rotate、skew,、translate 等）
- 尝试 3D 变换
- CSS3 动画效果（使用关键帧）

7.1 什么是 CSS3 过渡以及如何使用它

我们在给超链接设置样式时，一般都会设置一个悬停状态（hover）的效果，这种方法能明显地提醒用户他的鼠标指向的是一个超链接。虽然对越来越多的触摸屏设备没太大用

处，但不管怎么说，这种方法对网站和用户之间的交互是非常简单实用的。

通常，使用 CSS 时悬停状态就是一个开关。它默认有一个状态，然后在鼠标悬停时马上切换到另一种状态。但是使用 CSS3 过渡，正如其名字所暗示的，可以让元素从一种状态慢慢转换到另一种状态。这种转换并不局限于悬停状态，但可以从悬停讲起。

在上一章中，我们使用 CSS3 制作了一个有红色渐变背景的按钮。所使用的 CSS3 代码如下（为简洁起见省略了浏览器私有前缀）：

```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
    background-color: #b01c20;
    border-radius: 8px;
    color: #ffffff;
    padding: 3%;
    float: left;
    background: linear-gradient(90deg, #b01c20 0%, #f15c60 100%);
    margin-top: 30px;
    box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
    text-shadow: 0px 1px black;
    border: 1px solid #bfbfbf;
}
```

我们来给按钮增加一个悬停效果：

```
#content a:hover {
    border: 1px solid #000000;
    color: #000000;
    text-shadow: 0px 1px white;
}
```

两种状态下的效果如下，先看默认状态：



再看悬停状态：



7

这里只是在鼠标悬停时简单地修改了一下文字、文字阴影以及边框的颜色。所以，你可能想象出来，使用这段 CSS 代码，当鼠标悬停在按钮上面时，按钮会直接从第一种状态（白色文字）突变到第二种状态（黑色文字）——就是一个开关效果。我们来给第一段 CSS 规则添加一点 CSS3 魔法：

```
#content a {
    /*……原来的样式……*/
    transition: all 1s ease 0s;
}
```

现在再把鼠标悬停在按钮上，文字、文字阴影和边框阴影的颜色都会从第一种状态平滑过渡到第二种状态。注意，这里把过渡应用到了元素而不是悬停状态上。这样做是为了让元素的其他状态（如`:active`）也能设置不同的样式并拥有类似的效果。所以请记住，过渡声明要放在过渡效果开始的元素上。那过渡是怎么发生的呢？

7.1.1　过渡相关的属性

CSS3 过渡效果涉及四个属性，也可以使用包含这四个属性的缩写。

- `transition-property`：要过渡的 CSS 属性名称（比如 `background-color`、`text-shadow` 或者 `all`，使用 `all` 则过渡会被应用到每一个可能的 CSS 属性上）；
- `transition-duration`：定义过渡效果持续的时间（时间单位为秒，比如`.3s`、`2s` 或 `1.5s`）；
- `transition-timing-function`：定义过渡期间速度如何变化（比如 `ease`、`linear`、`ease-in`、`ease-out`、`ease-in-out` 或 `cubic-bezier`）；
- `transition-delay`：可选，用于定义过渡开始前的延迟时间。相反，将该值设置为一个负数，可以让过渡效果立即开始，但过渡旅程则会从半路开始。

单独使用各种过渡属性创建转换效果的语法如下：

```
#content a {
    ……其他样式……
    transition-property: all;
    transition-duration: 1s;
    transition-timing-function: ease;
    transition-delay: 0s;
}
```

1. 过渡的简写语法

正如我们之前所见过的那样，我们可以将单个的声明组合成一个简写版：

```
transition: all 1s ease 0s;
```

使用简写语法时要注意，声明中的第一个时间值总被应用给 `transition-duration`，第二个时间值总被应用给 `transition-delay`。

和以前一样，别忘了浏览器私有前缀。例如，上面那句简写声明添加了浏览器私有前缀之后，代码如下：

```
-o-transition: all 1s ease 0s;
-ms-transition: all 1s ease 0s;
-moz-transition: all 1s ease 0s;
-webkit-transition: all 1s ease 0s;
transition: all 1s ease 0s;
```

我们将没有前缀的标准版本放在了最后面，这样当浏览器完全实现了标准之后，这句代码就会覆盖之前带前缀的版本。

过渡的局限性

使用过渡时有一点需要说明，即有些属性无法实现过渡，尽管规范上说它应该可以（即使在最新的工作草案 http://dev.w3.org/csswg/css3-transitions/中也这么说）。例如，`background-gradient` 属性就无法过渡。但理论上所有 CSS 属性都是可以过渡的（http://www.w3.org/TR/css3-transitions/ #properties-from-css- ）。

2. 在不同时间段内过渡不同属性

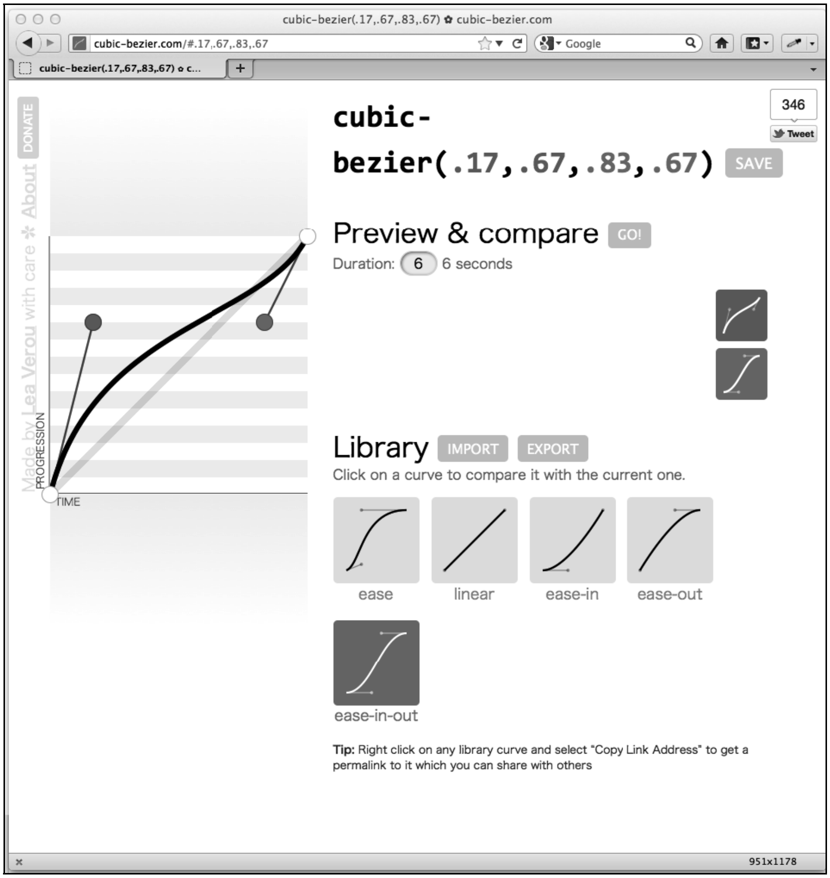
当一条规则要实现多个属性过渡时，这些属性不必步调一致。看看下面这段代码：

```
#content a {
    /*……其他样式……*/
    transition-property: border, color, text-shadow;
    transition-duration: 2s, 3s, 8s;
}
```

此处我们通过 `transition-property` 来指定只过渡 `border`、`color` 和 `text-shadow`，然后在 `transition-duration` 声明中我们设定边框过渡效果应该 2 秒内完成，文字颜色 3 秒，文字阴影 8 秒。由逗号分隔的过渡持续时间按顺序对应上面的 CSS 属性。

3. 理解过渡调速函数

大多数过渡属性都能一看就知道是什么意思。前面其实已经介绍了过渡相关的几个 CSS 属性。其中，过渡持续时间和延迟时间都是以秒为单位的时间值（如 2s），所以也很好理解。但过渡调速函数又有点让人摸不着头脑了。`ease`、`linear`、`ease-in`、`ease-out`、`ease-in-out` 以及 `cubic-bezier` 这些东西都是做什么用的？它们其实都是某种贝塞尔曲线，本质上就是缓动函数。我估计这样讲你还是不太明白。这样说吧……嗯，这其实就是那种用文字很难表达的概念（当然本人有很好的文字驾驭能力），就像你不得不给你的另一半解释清楚为什么你忘记了她的生日一样！所以我就不白费口舌了，还是请读者直接看这里吧 http://cubic-bezier.com/。



这个网站能让你对比查看各种调速函数，看到它们的区别。不过，即使你闭着眼睛都能写出贝塞尔曲线（而且作为老外还能用中国话从 1000 倒数到 1），在实际使用中，它们的效果也确确实实没有太大区别。为什么呢？

和任何增强效果一样，使用过渡效果也必须长个心眼儿。在现实当中，如果过渡效果持续的时间过长，会让网站感觉很慢。导航链接用整整 5 秒时间完成过渡，只会让你的用户骂娘而不是赞叹。因此，除非有什么特殊的理由，否则使用快速（以我的经验最多 1s）的默认过渡（ease）效果往往最好。

7.1.2　响应式网站中的有趣过渡

一旦变成响应式设计的发烧友，你就会发现自己在浏览网页时总要调整一下浏览器窗口大小，以此确认该网站是不是响应式的。但这种习惯可能会激怒“普通”人，所以最好私下里做这件事。

我经常访问的一个讨论 CSS 技术的好网站是 Chris Coyier 的 http://css-tricks.com。在它重新设计之后，我浏览时碰巧调整了浏览器窗口，在看到众多元素在屏幕上飞舞时，我露出了会心的微笑。Chri 给网页施了什么魔法使其具有了这种效果？其实代码如下[①]：

```
* {
    transition: all 1s;
}
```

此处，我们使用 CSS 通配选择器*来选择页面所有元素，然后为所有元素都设置一个耗时 1 秒的过渡效果。声明中省略了过渡调速函数，浏览器默认会使用 ease；声明中同样省略了延迟时间，浏览器默认使用 none，所以过渡效果不会有延迟。最终效果是什么样？大多数效果（超链接、悬停状态，等等）和你所期望的一样。不过，因为所有元素都被应用了过渡，自然也就包括媒体查询中的规则，所以当浏览器窗口大小发生变化时，页面元素将从一种排列方式过渡为另外一种排列方式。必须这么做吗？当然不是！但这种效果是不是既好看又好玩？没错！

7.2　CSS3 的 2D 变形

虽然两个英文单词发音相似，但 CSS 变形（transformation，包括 2D 变形和 3D 变形）和 CSS 过渡（transitions）完全不同。可以这样理解：过渡元素从一种状态平滑转换到另一种状态，而变形则定义了元素将会变成什么样子。我自己（极其幼稚）的理解是这样的：

> 想象一下《变形金刚》里的擎天柱，他先变形（transform）为别的东西，经过一段时间的过渡（transition）又变成了卡车。

可能刚才的这个比喻把你搞得更晕了（或者你根本不知道擎天柱是谁），还是直接说正题吧。我们来给 AND THE WINNER ISN'T 网站的导航链接在悬停时添加一个 2D 变形：

```
nav ul li a:hover {
    transform: scale(1.7);
}
```

在现代浏览器中，在导航链接上悬停鼠标就会看到如下效果：



① 该网站最新版的 CSS 代码中没有这句代码，所有元素都应用过渡效果，影响性能。——译者注

我们告知浏览器当鼠标悬停在链接上时，将其放大到原始大小的 1.7 倍。

现在如果你尝试在 Safari 浏览器中添加这种规则，须注意它需要应用该规则的原始元素必须以块状显示。示例如下：

```
nav ul li a {
    height: 42px;
    text-decoration: none;
    text-transform: uppercase;
    color: black;
    text-shadow: 0 1px 0 hsla(0, 0%, 100%, 1.0);
    font: 1.875em/42px 'BebasNeueRegular';
    display: block;
}
```

不这样做的话，什么效果都没有，你懂的，垃圾得很。

我们能做哪些变形

有两种可用的 CSS3 变形：2D 变形和 3D 变形。2D 变形的实现更广泛，浏览器支持更好，写起来也更简单些，所以我们先来看看 2D 变形。CSS3 的 2D 变形模块允许我们使用下列变形。

- `scale`：用来缩放元素（放大或缩小）
- `translate`：在屏幕上移动元素（上下左右四个方向）
- `rotate`：按照一定角度旋转元素（单位为度）
- `skew`：沿 X 和 Y 轴对元素进行斜切
- `matrix`：允许你以像素精度来控制变形效果

接下来我们逐个试验一下，看看会有什么效果。

1. scale

我们前面看到过这种变形效果。但是除了我们刚才使用的正整数值，还要知道使用小于 1 的值可以缩小元素；下面的代码就会将元素缩小一半：

```
transform: scale(0.5);
```



2. translate

```
transform: translate(40px, 0px);
```

translate 会告知浏览器按照一定度量值移动元素，可以使用像素或百分比。语法是第一个值表示从左向右移动的距离（本例中是 40 像素），然后是从上往下移动的距离（本例中是 0 像素，所以元素与其他导航链接保持对齐）。正值会让元素向右或向下移动，负值则会让元素向左或向上移动。所以本例中对导航链接的声明效果是：链接在鼠标悬停时水平向右移动 40 像素。



3. rotate

```
transform: rotate(90deg)
```

rotate 允许你旋转一个元素。在本例中，我们将链接悬停的变形修改为旋转 90 度。在浏览器中的效果如下：



括号中的值只能以度为单位（如 90deg）。当然，这也挡不住你胡来——你可以让一个元素按照下面所设定的这个值来旋转：

```
transform: rotate(3600deg);
```

这会让元素旋转整整 10 圈。这么奇怪的值，其用处少之又少，除非你给一家风车公司设计网站，那倒可能会派上用场。

4. skew

如果你多少有点 Photoshop 经验，就会知道 skew（斜切）是怎么回事，它会让元素在一个或两个轴上变形偏斜。

```
transform: skew(10deg, 2deg);
```

在悬停状态的导航链接上应用该规则，产生的效果如下：



第一个值是 X 轴上的斜切（本例中是 10 度），第二个值是 Y 轴上的斜切（本例中是 2 度）。省略第二个值意味着仅有的值只会应用在 X 轴上（水平方向）。例如：

```
transform: skew(10deg);
```

这样写完全有效，只是斜切仅会作用于 X 轴。斜切值始终以度为单位。正值沿顺时针方向斜切，负值则沿逆时针方向斜切。

5. matrix

好，现在该聊聊那个被严重抬高的电影[①]了。什么电影？！别忘了你要讲的是 CSS3 中的 `matrix`，不是电影！是吗？好吧，看下面……

`matrix` 变形的语法看起来超复杂：

```
transform: matrix(1.678, -0.256, 1.522, 2.333, -51.533, -1.989);
```

它基本上能让你将若干变形效果（scale、rotate、skew 等等）组合成单个声明。上面的声明在浏览器中产生的效果如下：

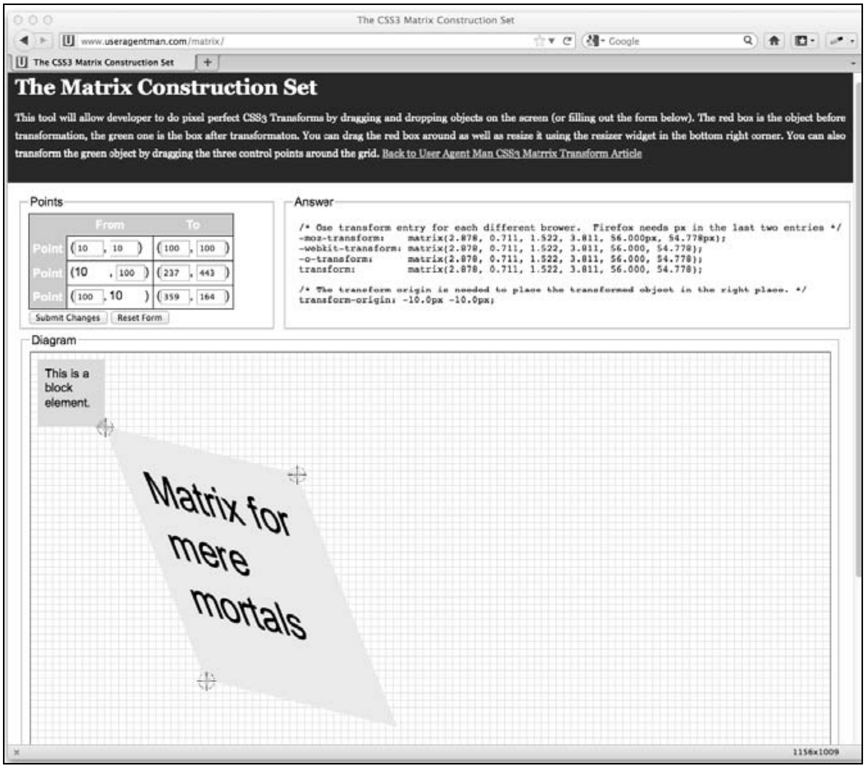


总的来说，我还是蛮喜欢挑战难题的，但我相信你也会同意这语法也太有挑战性了。而且在你看了规范文档之后就会发现问题更难，要完全理解矩阵你得了解相关的数学知识：http://www.w3.org/TR/css3-2d-transforms/#cssmatrix-interface。

傻瓜化的矩阵变形工具

无论怎么想象我都不是一个数学家，所以当我需要创建矩阵变形时，我一般都走捷径。如果你的数学也不太好，我建议你访问这里：http://www.useragentman.com/matrix/。

① Matrix 即电影《黑客帝国》，详见 http://baike.baidu.com/view/180529.htm#sub5119498。——译者注



Matrix Construction Set 这个网站可以让你精确地将元素拖曳成想要的样子，然后它会自动生成完美的矩阵代码（代码中包含了浏览器私有前缀）。

6. transform-origin 属性

在使用上述变形效果的同时，你还可以使用 `transform-origin` 属性来修改变形效果的起点：

```
transform: rotate(45deg);
transform-origin: 20% 20%;
```

将上面的规则应用到导航链接上，鼠标悬停之后的效果如下：



`transform-origin` 属性默认就在起作用，变形的起点默认是元素的中心点。这个属性提供了一种方便的方法来移动变形的中心点，这样能做出很多很赞的效果。

transform-origin 属性的详细信息请见这里：http://www.w3.org/TR/css3-2d-transforms/#transform-origin-property。

以上讲述了 2D 变形的基本要素。比起 3D 变形，2D 变形在浏览器中被广泛地支持，它为我们提供了一种轻量便捷的方法，让现代浏览器的用户体验得以锦上添花。

CSS3 的 2D 变形模块的完整规范文档请见：http://www.w3.org/TR/css3-2d-transforms/。

7.3 尝试 CSS3 的 3D 变形

Webkit 核心浏览器（Safari 和 Chrome）和 Firefox 10+都已支持 CSS3 的 3D 变形，但最新的 IE10 都还不支持该特性。尽管缺少“桌面版”浏览器的广泛支持，但多亏了移动平台浏览器基本都是 Webkit 血统，所以 3D 变形在 Android（V3 以后的版本）和 iOS（所有版本）上均被支持。

从这点上来讲，你最好在 Webkit 核心浏览器（如 Chrome 或 Safari）中测试网页效果（当然，也可以选择其他支持 3D 变形的浏览器）。

现在我们开始尝试 3D 变形。这是一片广阔天地，有几乎无穷的可能性。我想象着有一天 3D 变形被广泛支持，我们就可以用它来做图片轮播效果，而不用再依赖 jQuery 等 JavaScript 方案了。不过，在那一刻到来之前，我们先来体验一下。

假设给 AND THE WINNER ISN'T 网站做一个简单测验功能。测试题目由一组电影海报组成，你必须猜猜世界上最权威的电影评论家（哈哈，说的就是我！）对这些电影的评判是好是坏。将鼠标悬停在图片上（在触摸屏上则是触击图片）就能揭晓答案。

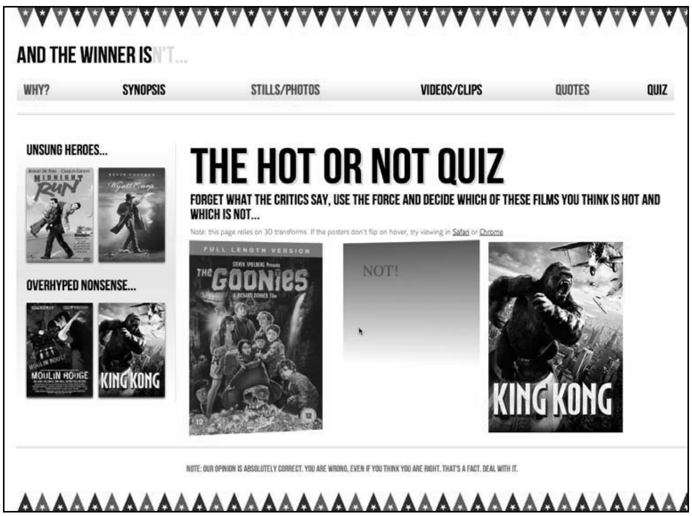
下面的代码是相关的页面标签，其中省略了重复的图片标签，因为它们的结构都一样：

```
<section class="Qcontainer">
  <div class="film">
    <div class="face front">
      <img src="img/goonies.jpg" alt="The Goonies" />
    </div>
    <div class="face back"><h5>HOT!</h5></div>
  </div>
</section>
```

对应的 CSS 代码如下。注意，因为 Webkit 浏览器对 3D 变形的支持比较好，所以下面的声明都使用特定的浏览器前缀。和之前一样，在实际开发中，浏览器私有前缀就是你形影不离的好朋友。

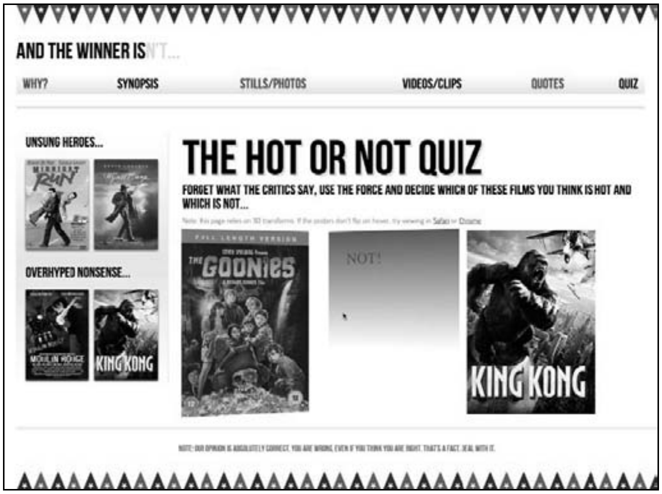
```
.Qcontainer {
    height: 100%;
    width: 28%;
    position: relative;
    -webkit-perspective: 800;
    float: left;
    margin-right: 2%;
}
.film {
    width: 100%;
    height: 15em;
    -webkit-transform-style: preserve-3d;
    -webkit-transition: 1s;
}
.Qcontainer:hover .film {
    -webkit-transform: rotateY(180deg);
}
.face {
    position: absolute;
    -webkit-backface-visibility: hidden;
}
.back {
    width: 66%;
    height: 127%;
    -webkit-transform: rotateY(180deg);
    background: #3b3b3b;
    background: -webkit-linear-gradient(top,
                            rgba(0,0,0,0.65) 0%,
                            rgba(0,0,0,0) 100%);
    padding: 15%;
}
```

代码部署好之后，将鼠标悬停在海报图片上，会看到图片翻转到背面并显示出了该电影的评判结果。



7.3.1　分析 3D 变形效果

我们来研究一下代码，看看 3D 变形效果是如何实现的。



第一个要点是在父级元素上设置透视。这样就开启了 3D 场景：

```
.Qcontainer {
    height: 100%;
    width: 28%;
    position: relative;
    -webkit-perspective: 200;
    float: left;
    margin-right: 2%;
}
```

透视的值越大，就表示你的视点与 3D 场景之间的景深越大。因此，如果想要一点隐约的 3D 效果，就增大透视值；如果想要非常明显的 3D 效果，则减小透视值。（3D 效果的立体程度，取决于 3D 场景与观察者之间的距离。）

下一个要点：

```
.film {
    width: 100%;
    height: 15em;
    -webkit-transform-style: preserve-3d;
    -webkit-transition: 1s;
}
```

`.Qcontainer` 类中添加的透视声明只会应用到其第一个子元素上（即本例中的 `class` 为`.film` 的 div）。因此，为了延续父元素的透视，我们给`.film` 元素设定了 `preserve-3d`（这样可以设置一个 3D 场景）。

接下来，当鼠标悬停在`.Qcontainer`模块上时，我们给`.film`这个 div 添加一个翻转效果：

```
.Qcontainer:hover .film {
    -webkit-transform: rotateY(180deg);
}
```

接下来的规则用来处理当海报翻转之后隐藏在其背面内容：

```
.face {
    position: absolute;
    -webkit-backface-visibility: hidden;
}
```

`.face`必须使用绝对定位，这样海报才能盖在`.back`这个 div 的上面：

```
.back {
    width: 66%;
    height: 127%;
    -webkit-transform: rotateY(180deg);
    background: #3b3b3b;
    background: -webkit-linear-gradient(top,
                           rgba(0,0,0,0.65) 0%,
                           rgba(0,0,0,0) 100%);
    padding: 15%;
}
```

最后，我们给`.back`这个 div 也加上`rotateY`。不加这句的话，`.back`这个 div 就会显示在正面海报之上。

这就全部做完了。现在将鼠标移到任意一张海报上都会看到超级酷炫的 3D 效果。

不过，对于非 Webkit 核心浏览器，页面效果肯定就残废了：



还好，我们可以用一点传统的 CSS 代码为非 Webkit 核心浏览器提供一个合理的降级方案：

```
.front {
    z-index: 5;
}
.Qcontainer:hover .front {
    z-index: 0;
}
```

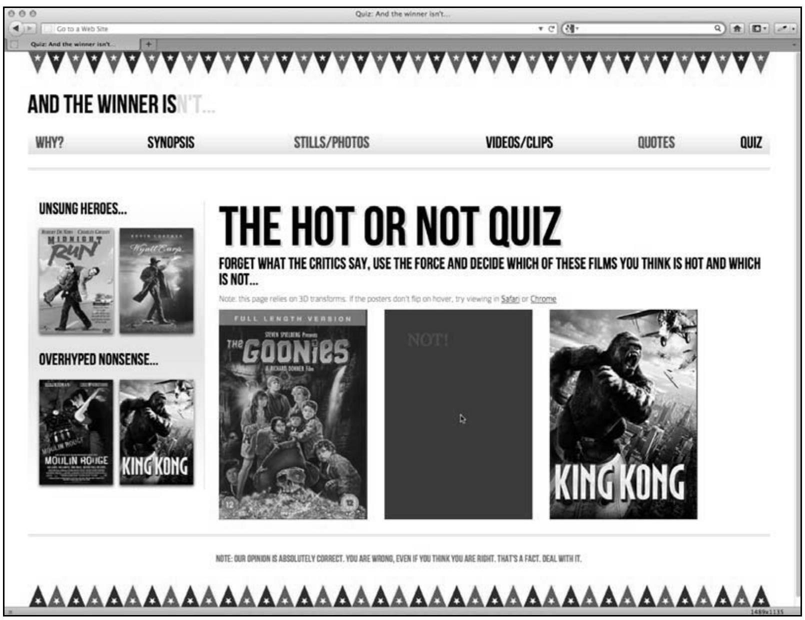
首先，我默认给.front 设置了 z-index 为 5，以便让其显示在.back 上面：

```
.front {
    z-index: 5;
}
```

然后，当.Qcontainer 模块处于悬停状态时，我们给.front 设置 z-index 为 0 以便让其隐藏在.back 后面：

```
.Qcontainer:hover .front {
    z-index: 0;
}
```

这样我们就给不支持 3D 变形的浏览器也提供了简单问答功能，只是缺少优雅的 3D 效果。



7.3.2　3D 变形尚未成熟

根据我的实践经验，目前大多数 3D 变形在使用百分比尺寸时效果都不太好（例如，修改

上节例子中的视口宽度，将会使 3D 变形效果完全错乱）。所以现在要让 3D 变形在响应式布局中呈现完美效果还有一些问题。此外，由于浏览器的支持有限，在制作跨浏览器网站时，3D 变形无法提供健壮的解决方案。目前，我依然使用 jQuery 或类似技术来制作这类变形效果。

但是，CSS3 3D 变形的前途是光明的，当浏览器开始广泛支持时，它就为我们提供了一次难得的机遇，让我们可以将现在依赖于 JavaScript 的优雅效果移植到样式表中。

> 想了解 W3C 有关 CSS 3D 变形的最新进展，请见这里：http://dev.w3.org/csswg/css3-3d-transforms/。

7.4　CSS3 动画效果

如果你制作过 Flash 动画，那 CSS3 动画你也能立即上手。CSS3 沿用了在 Flash 和其他基于时间线的应用程序中被广泛使用的动画关键帧技术。

相较于 3D 变形，CSS3 动画的浏览器支持度更高。Firefox 5+、Chrome、Safari 4+、Android (所有版本)、 iOS (所有版本)均支持，IE 10 也决定加入该行列。

CSS3 动画由两部分组成：首先是关键帧声明，然后在动画属性中使用该关键帧声明。

上一节我们制作了一个简单的翻转效果，用于展示我对电影的评判结果。但是翻转效果的背面文字非常丑陋，所以我们来给这些文字添加一个有趣的闪烁效果。

首先是关键帧规则：

```
@keyframes warning {
  0% {
    text-shadow: 0px 0px 4px #000000;
  }
  50% {
    text-shadow: 0 0 20px #000000;
  }
  100% {
    text-shadow: 0px 0px 4px #000000;
  }
}
```

此处的代码没有加前缀，如果在浏览器中没有效果，你可能需要添加一组完整的浏览器私有前缀（如`@-webkit-keyframes`）。

我们来分析一下上面的代码：

```
@keyframes warning {
  0% {
    text-shadow: 0px 0px 4px #000000;
```

```
    }
    50% {
      text-shadow: 0 0 20px #000000;
    }
    100% {
      text-shadow: 0px 0px 4px #000000;
    }
}
```

首先，我们定义了一个@keyframes（关键帧）声明。然后为这个特定的关键帧声明设置了一个名称：warning。你可以将其叫成任何名字，但考虑到这些关键帧声明可以在多个元素上复用，所以建议取一个合理的名字。

可以设置多个关键帧（比如百分比值 10%、20%、30%、40%等等），或者也可以使用 from 和 to 值来定义动画的开始帧和结束帧。但注意 Webkit 浏览器在使用 from 和 to 值的情况下效果无法保证（它更喜欢 0%和 100%）。

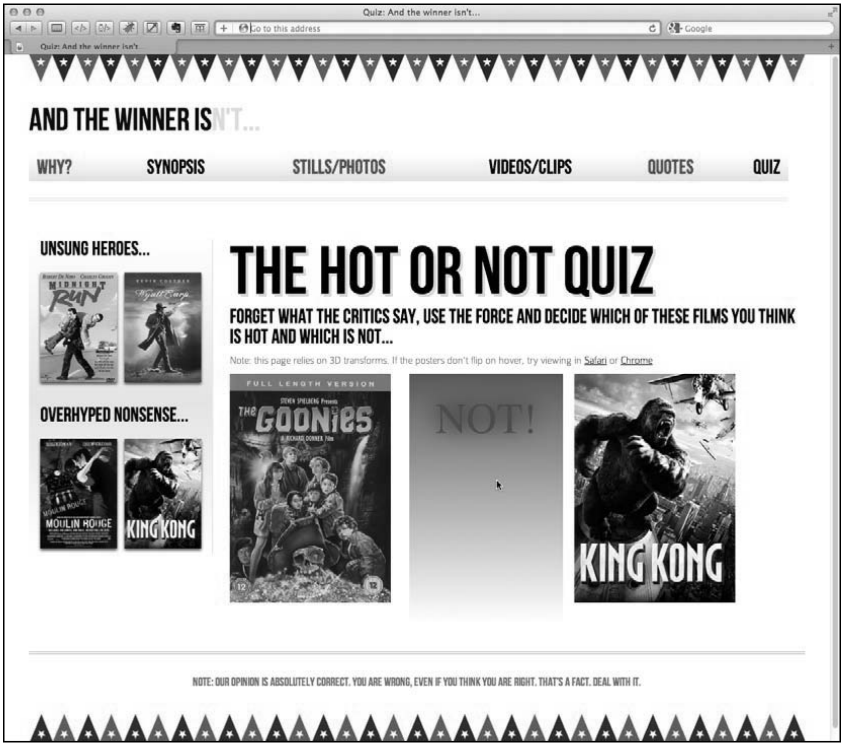
```
@keyframes warning {
  from {
    text-shadow: 0px 0px 4px #000000;
  }
  50% {
    text-shadow: 0 0 40px #000000;
  }
  to {
    text-shadow: 0 0 4px #000000;
  }
}
```

本例给文字阴影加一点动画，动画开始时是 4 像素阴影，然后用 50%的时间变化至 40 像素阴影，之后再变化回 4 像素阴影。

我们已经声明了关键帧，接下来可以在动画属性中引用它：

```
.back h5 {
    font-size: 4em;
    color: #f2050b;
    text-align: center;
    animation: warning 1.5s infinite ease-in;
}
```

在 animation 属性之后，我们设定了要使用的关键帧（例子中的 warning），然后设定了动画的持续时间（1.5s），之后设定了 animation-iteration-count（我们在此时使用了 infinite 让动画连续循环播放），最后设定了调速函数（ease-in）。静态截图显然无法展示出动画，但希望你能想象得出文字阴影一闪一闪的效果。访问这里查看效果：http://www.andthewinnerisnt.com。



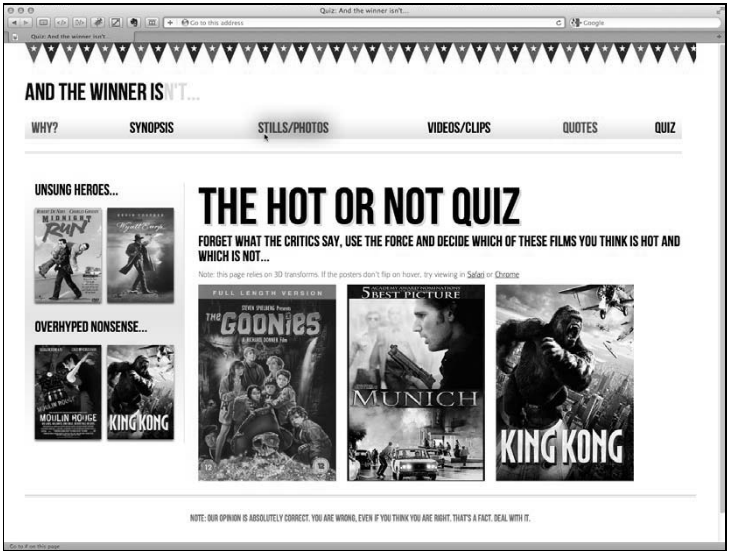
CSS3 动画缩写语法可以接受全部 7 个独立动画属性的值。除了上例中使用的之外，还可以设定 `animation-delay`（动画开始前的延时），`animation-play-state`（值可以是 `running` 或 `paused`，用于控制动画的播放和暂停），最后还有 `animation-fill-mode`，这个属性到现在我也没找到用武之地（默认值是 `none`）。当然也可以不用缩写语法，而是像下面这样将它们一个一个单独列出来：

```
animation-name: warning;
animation-duration: 1.5s;
animation-timing-function: ease-in-out;
animation-iteration-count: infinite;
animation-play-state: running;
animation-delay: 0s;
animation-fill-mode: none;
```


前面已经提过，在其他元素上复用动画效果非常简单，如下所示：

```
nav ul li a:hover {
    animation: warning 1.5s infinite ease-in;
}
```

这样导航链接也会有同样的闪烁效果。你可以（希望能）看到下面截图中的 STILLS/PHOTOS 链接正处在闪烁过程中。访问这里看看真实效果：http://www.andthewinnerisnt.com。



这只是使用 CSS3 动画的一个简单示例。事实上任何 CSS 属性都可以用在关键帧动画中，所以有无限可能性。网上有数不清的 CSS3 动画技巧范例，像 http://webdesignerwall.com/trends/47-amazing-css3-animation-demos 这个网页就足以给你提供丰富的 CSS3 动画灵感。

> 想了解 W3C 有关 CSS 动画的最新进展，请见这里：http://dev.w3.org/csswg/css3-animations/。

组合使用 CSS3 变形和动画

我们来尝试再做一种效果，秀一下我们的 CSS3 功力。我想将侧边栏的图片按照不同的角度来放置，然后让它们动起来。目标就是当页面刚加载进来的时候这些图片都“晃”几下。侧边栏的标签代码如下：

```
<aside>
  <div role="complementary">
    <div class="sideBlock unSung">
      <h1>Unsung heroes...</h1>
      <a href="#"><img src="img/midnightRun.jpg"
                       alt="Midnight Run" /></a>
      <a href="#"><img src="img/wyattEarp.jpg"
                       alt="Wyatt Earp" /></a>
    </div>
  </div>
  <div role="complementary">
    <div class="sideBlock overHyped">
      <h1>Overhyped nonsense...</h1>
      <a href="#"><img src="img/moulinRouge.jpg"
```

```
                    alt="Moulin Rouge" /></a>
      <a href="#"><img src="img/kingKong.jpg"
                    alt="King Kong" /></a>
    </div>
  </div>
</aside>
```

接下来我们开始编写相关的 CSS3 代码。首先，创建一个名为 swing 的关键帧声明：

```
@-webkit-keyframes swing {
  from {
    transform: rotate(3deg);
  }
  20% {
    transform: rotate(7deg);
  }
  60% {
    transform: rotate(10deg);
  }
  80% {
    transform: rotate(7deg);
  }
  to {
    transform: rotate(3deg);
  }
}
```

这个动画使用了 2D 旋转变形，以使图片从 3 度旋转至 10 度然后再旋转回原位。接下来给图片上追加 animation 属性：

```
#quiz .unSung a:nth-child(odd) img {
    transform: rotate(3deg);
    animation: swing 0.1s 5 ease-in;
}
#quiz .unSung a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.3s ease-in;
}
#quiz .overHyped a:nth-child(odd) img {
    transform: rotate(3deg);
    animation: swing 0.1s 5 0.2s ease-in;
}
#quiz .overHyped a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.5s ease-in;
}
```

我们来分析一下上面的代码。首先我们使用了 CSS 的 id 选择器，这样就可以将该规则限定应用在 QUIZ 页面上（该页面上的 body 标签是<body id="quiz">）。

在添加动画属性之前，我想先给图片设置一个默认的旋转变形，以便让它们在动画结束之后仍然是倾斜的。但我不想它们都按同一个角度倾斜，所以我们使用从第 5 章中学来

7

的 nth-child 选择器来分别选择出奇数与偶数图片，然后应用不同的旋转变形：

```
#quiz .unSung a:nth-child(odd) img {
    transform: rotate(3deg);
    animation: swing 0.1s 5 ease-in;
}
#quiz .unSung a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.3s ease-in;
}
#quiz .overHyped a:nth-child(odd) img {
    transform: rotate(3deg);
    animation: swing 0.1s 5 0.2s ease-in;
}
#quiz .overHyped a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.5s ease-in;
}
```

然后给每张图片都追加 animation 属性，注意规则之间的差别。缩写语法会将第二个时间值（0.5 秒）认定为动画延时时间。通过设定这个值，就可以分别在不同的时间启动各个动画效果。

```
#quiz .overHyped a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.5s ease-in;
}
```

同样，再好的文字也展示不出来动画效果。如果你这会儿上不了网，那我只能告诉你，侧边栏的电影海报会快速地左右晃动几下，之后就会像下图所示那样倾斜地摆放着：



7.5 小结

有关 CSS3 过渡、变形和动画的内容，足足可以写好几本书。但是我希望通过本章蜻蜓点水般的学习，你能了解它们的基本知识并让它们运行起来。其实，我们采用 CSS3 新特性和技巧的最终目的，是想使用 CSS3 来替代 JavaScript 制作一些优雅精美的增强效果，从而使响应式设计的代码更加简洁、效果更加丰富。本章我们学习了 CSS3 过渡是什么，以及如何编写相应代码，了解了 `ease`、`linear` 等调速函数，并在响应式设计中应用它们制作了简单但有趣的效果。还学习了缩放、斜切等 2D 变形，以及如何将它们与过渡组合使用。此外，还在学习 CSS3 动画的强大功能与简洁语法之前，简单了解了 3D 变形。你要相信自己的 CSS3 功力正在不断增长！

但是，如果网站设计中有一个领域是我尽可能避免谈及的（就如我拼命不想看《慕尼黑》或《金刚》一样），那肯定就是表单。我不知道为什么，只是一直觉得制作表单是件单调乏味的事儿。当我知道了 HTML5 和 CSS3 可以让表单的制作、美化甚至验证（没错，就是验证！）过程都比以往简单很多时，简直把我给乐坏了。我高兴得都手舞足蹈了，相信到时候你也会跟我一样。下一章我们就来学习 HTML5 表单。

7

用 HTML5 和 CSS3 征服表单 8

过去，想要让表单风格跨浏览器保持一致是很困难的。而且还需要 JavaScript 来验证表单输入，此外，也缺少一些处理日常信息（如电话号码、电子邮箱以及 URL）的输入类型。

好消息是 HTML5 基本上解决了这些常见的问题。接下来我们来熟悉一下新的 HTML5 表单特性，来看看它们如何减轻我们的负担。

在响应式设计中使用 HTML5 制作表单会带来额外的好处。它能让我们再次简化代码，为用户提供尽可能简洁的页面。对于不支持这些新特性的浏览器，我们有工具给它们打补丁，也能做到表现一致。

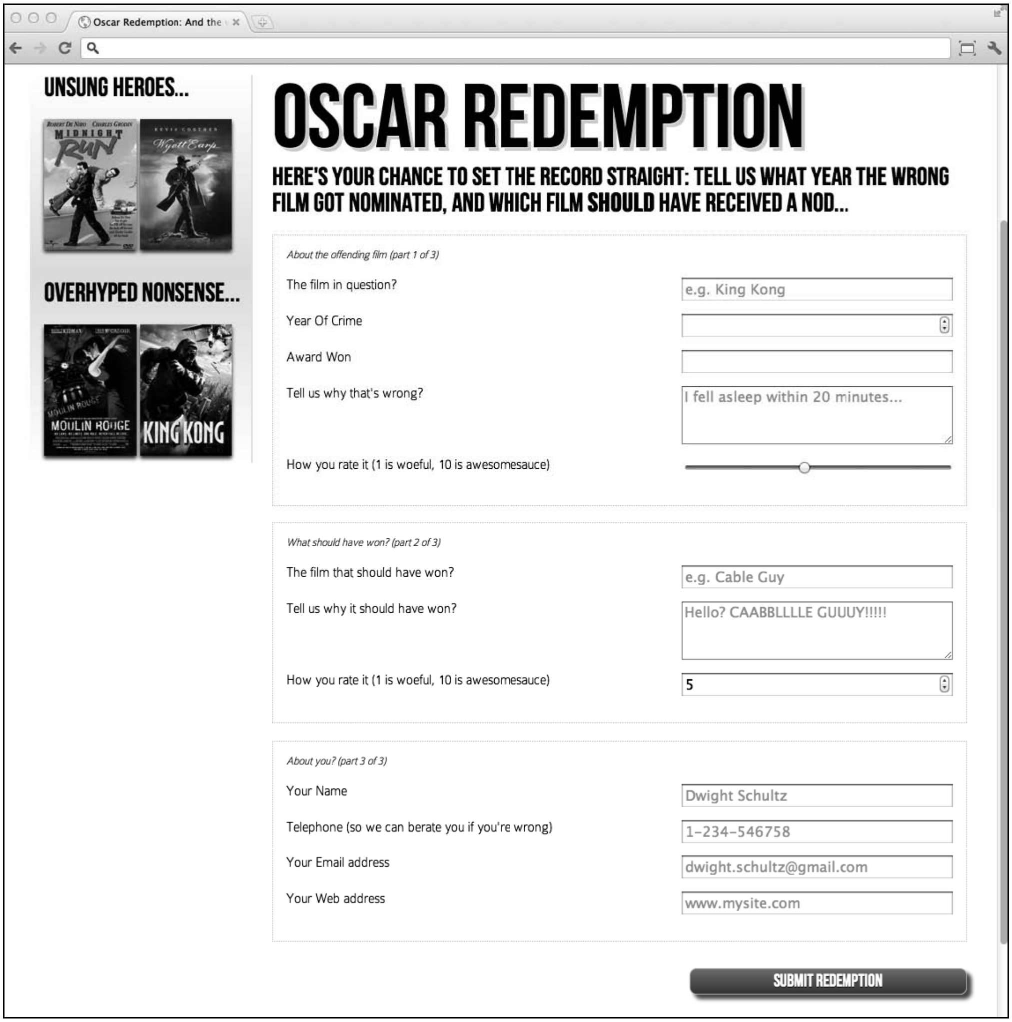
本章内容

- 在表单域中插入占位符文字
- 在需要时禁用表单域的自动完成功能
- 将特定表单域设置为必填项
- 使用不同类型的输入框，如电子邮箱、电话号码和 URL。
- 制作数字输入滑动条以方便选择数值
- 使用日期和颜色选择器
- 学习如何使用正则表达式定义表单值验证规则
- 为落后的浏览器添加腻子脚本
- 使用 CSS3 轻松灵活地美化 HTML5 表单

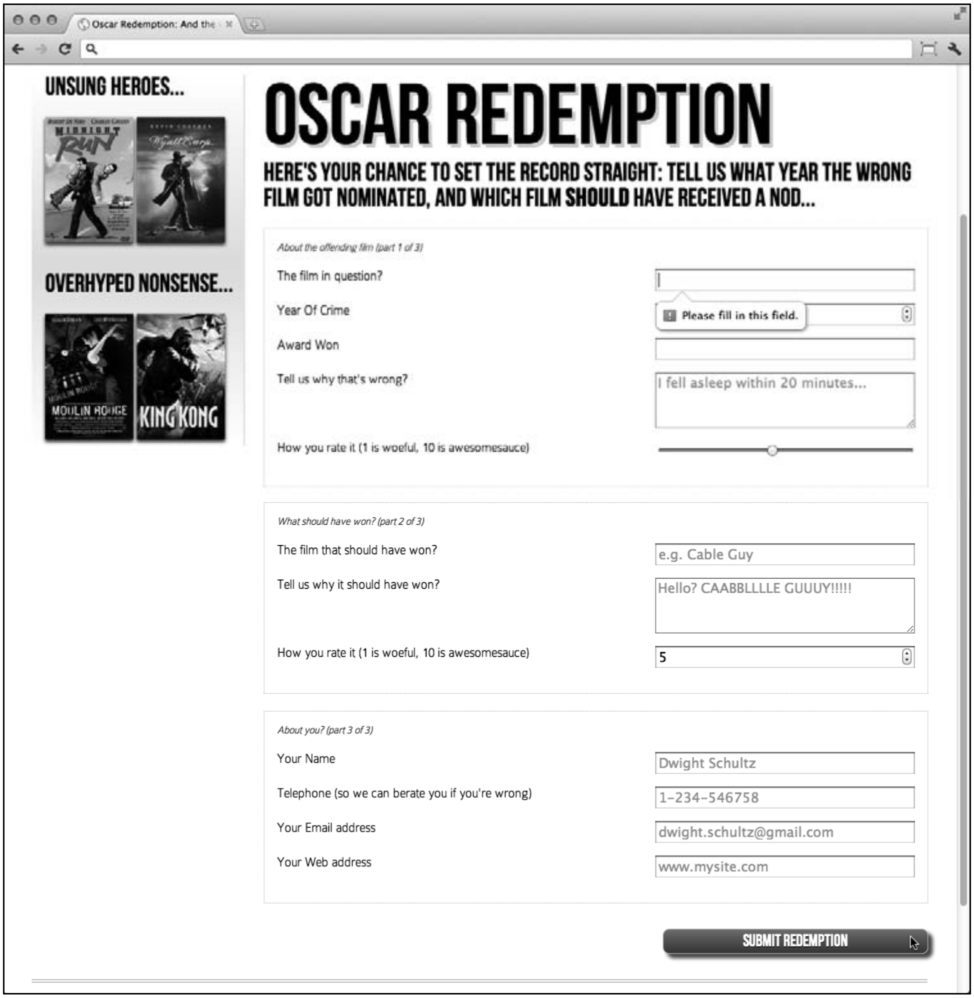
8.1 HTML5 表单

我决定让人们可以在网站上对那些已经获得大奖的烂电影发泄他们的不满。我要制作一个表单，让人们发表想法，说说哪些电影不该获奖，而那些电影应该得奖。

下图展示了我们要制作的表单在 Chrome（v16）中添加了一点基本样式后的效果：



除了标准的表单输入框和文本输入域之外，表单中还有一个数字控制器、一个滑动输入条，并且很多表单域都有占位符文字。如果我们聚焦（选中）某个表单域，其中的占位符文字就会自动消失；如果表单域失去焦点且未输入任何内容（点击一下输入框以外的区域即可），占位符文字又会再次显示。此外，如果在 Google Chrome 浏览器中查看页面，当我们准备提交没有任何内容的表单时，就会看到下面的结果：



页面不仅包含一些视觉元素（滑动条和控制器），还具有基本的客户端验证功能。我们知道，过去的表单验证一般都需要依赖 JavaScript。

现在，所有这些用户界面元素（包括上述的滑动条、占位符文字和控制器）和表单验证都可以使用原生的 HTML5 来完成，不需要再依赖 JavaScript，这是个很好的消息。让我们来看看如何使用这些新的表单特性。

8.1.1　理解 HTML5 表单中的元素

我们的 HTML5 表单包含很多元素，所以我们拆开来讲。首先表单被设定了一个 ID 用于对应样式，然后包含一个 HTML5 的 `hgroup`，用于显示表单标题和说明文字：

```
<form id="redemption" method="post">
  <hgroup>
    <h1>Oscar Redemption</h1>
    <h2>Here's your chance to set the record straight: tell us what
      year the wrong film got nominated, and which film <b>should</b>
      have received a nod...</h2>
  </hgroup>
```

表单中的三块子区域都使用带有 `legend` 标签的 `fieldset` 来包裹：

```
<fieldset>
<legend>About the offending film (part 1 of 3)</legend>
<div>
  <label for="film">The film in question?</label>
  <input id="film" name="film" type="text" placeholder="e.g. King
    Kong" required aria-required="true" >
</div>
```

从上面的代码片段可以看出，每一个输入元素，都有一个 `label` 元素与之对应，且一并被包裹在 `div` 中，看起来是很普通的表单。不过，此处的第一个输入元素中使用了一个 HTML5 的表单特性：在 `id`、`name` 和 `type` 这些普通属性之后，有一个 `placeholder` 属性。

8.1.2 placeholder

`placeholder` 属性看起来类似这个样子：

```
placeholder="e.g. King Kong"
```

我们常常需要在表单域中显示占位符文字,所以 HTML5 的设计者决定让浏览器内建标签对其提供支持。只需在 `input` 元素中加入 `placeholder` 属性，其属性值就会默认显示为占位符文字，输入框获取焦点时该文字自动消失。当输入框失去焦点且没有任何输入值时，占位符文字则会再次显示。

上节代码片段的 `placeholder` 属性之后，是另一个 HTML5 表单特性 `required` 属性。

8.1.3 required

`required` 属性看起来是这个样子的：

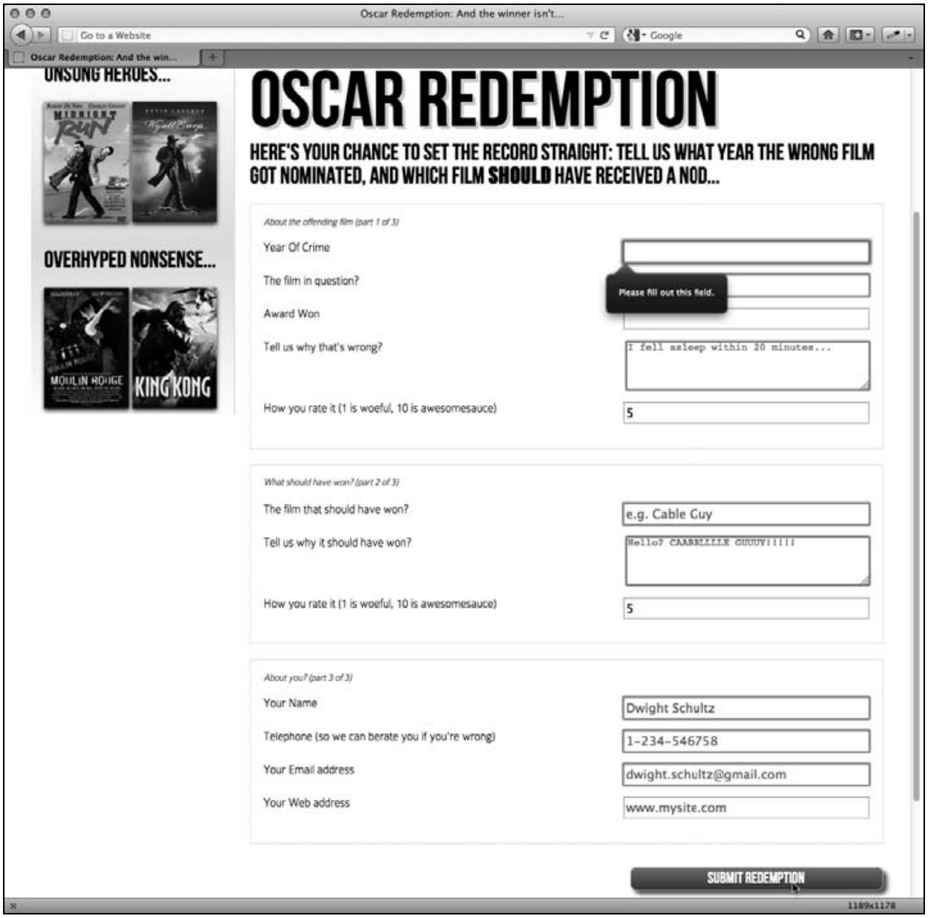
```
required aria-required="true"
```

在支持 HTML5 的浏览器中，在 `input` 元素中追加布尔类型的属性 `required`（也就是说你可以选择追加或不追加该属性），则表明该表单域为必填项。如果表单提交时该必填项没有任何信息，浏览器则会显示警告信息。警告信息的显示方式（包括显示内容和样式）取决于浏览器与输入框类型。除了 HTML5 的 `required` 属性，我们的例子中还追加了一个等价的 WAI-ARIA 属性：`aria-required="true"`。如果没有特殊原因，建议

8

为输入框追加 WAI-ARIA 版的必填属性以方便屏幕阅读器用户（你应该还记得，我们在第 4 章学习过 WAI-ARIA 技术）。

我们之前已经看过 Chrome 中必填项警告信息的效果，下面的截图展示了 Firefox 中的效果：



`required` 属性可用于多种类型的输入元素，以确保表单域必须输入值。`range`、`color`、`button` 和 `hidden` 类型的输入元素则不能使用 `required`，因为这几种输入类型几乎都有默认值。

另一个可追加到输入元素上的 HTML5 表单属性是 `autofocus`。

8.1.4　autofocus

HTML5 的 `autofocus` 属性可以让表单在加载完成时就有一个表单域被默认聚焦（或选

中），以便于用户输入。下面的代码中被 div 包裹的表单域的末尾就被追加了一个 autofocus 属性：

```
<div>
  <label for="search">Search the site...</label>
  <input id="search" name="search" type="search" placeholder="Wyatt Earp"
autofocus>
</div>
```

使用该属性时要小心。如果有多个表单域都被追加了 autofocus 属性，则会造成跨浏览器混乱。例如有多个表单域追加了 autofocus 属性，在 Chrome（v16）中会聚焦最后一个使用 autofocus 属性的表单域，而 Firefox（v9）恰恰相反，会聚焦第一个使用 autofocus 的表单域。

还有一点需要注意，有些用户会使用空格键让网页内容向下滚动。如果网页的表单中含有带 autofocus 属性的表单域，则会阻止空格键的默认行为，这时敲击空格键会向已聚焦的输入框中输入空格。显然，这会让用户很懊恼。

8.1.5 autocomplete

很多浏览器默认提供自动完成功能以帮助用户输入。以往用户可以在浏览器设置中打开或关闭这项功能，现在我们还能告知浏览器我们不想在某个表单或表单域上使用自动完成功能。这不仅仅能保护敏感数据（例如银行账户），还可以让你确保用户用心填写表单，手工输入一些值。比如以前在填写表单时，如果需要填写电话号码，我就输入一个假号码。我保证不止我一个人这么做（大家不都是填假号码嘛？），但我也能保证如果在相关的输入项上禁用自动完成功能，那用户肯定不会输入一个假号码。下面的代码演示了一个禁用自动完成功能的表单项：

```
<div>
  <label for="tel">Telephone (so we can berate you if you're  wrong)</label>
  <input id="tel" name="tel" type="tel" placeholder="1-234-546758"
autocomplete="off" required aria-required="true" >
</div>
```

我们也可以通过给表单本身（不是 fieldset）设置属性来禁用整个表单的自动完成功能。示例代码如下：

```
<form id="redemption" method="post" autocomplete="off">
```

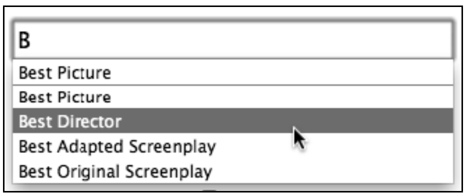

8.1.6 list（及对应的 datalist 元素）

list 属性以及对应的 datalist 元素可以让用户在输入框中开始输入值的时候，显示一组备选值。下面是一个包含在 div 中的使用 list 属性及对应 datalist 元素的代码示例：

```
<div>
  <label for="awardWon">Award Won</label>
  <input id="awardWon" name="awardWon" type="text" list="awards">
  <datalist id="awards">
    <select>
      <option value="Best Picture"></option>
      <option value="Best Director"></option>
      <option value="Best Adapted Screenplay"></option>
      <option value="Best Original Screenplay"></option>
    </select>
  </datalist>
</div>
```

`list` 属性中的值（`awards`）同时也是 `datalist` 元素的 `id`。这样就可以让 `datalist` 与输入项关联起来。虽然将 `option` 包裹在 `select` 中不是必需的，但这样做便于为老版本浏览器提供降级方案。

使用了 list 属性的输入框看起来就是一个普通的文本输入框，当开始输入时，输入框下面就会显示一个数据选择框，其中包含从 datalist 中检索到的匹配数据。在下面的截图中可以看到效果（Firefox v9）。在本例中，因为 datalist 中的所有 option 都含有 B，所以所有数据都显示出来了。



但当输入 D 的时候，就只有匹配的数据才被显示出来，效果如下图：



`list` 输入框不会阻止用户输入自己想输入的内容，不过它为我们提供了一种便利的方式来添加输入提示功能，增强用户体验。

8.1.7 HTML5 的新输入类型

HTML5 新增了很多输入类型，它们最大的作用就是可以限制用户输入的数据，不需要引入额外的 JavaScript 代码。这些新的输入类型最赞的一点，就是在那些不支持新特性的浏览器中，它们会被降级显示为一个标准的文本输入框。此外还有很多很有用的赋予脚本

可以让那些老版本浏览器也跟上时代。我们稍后会讲到这些内容，现在先来看看这些新的 HTML5 输入类型以及它们所带来的便利。

1. email

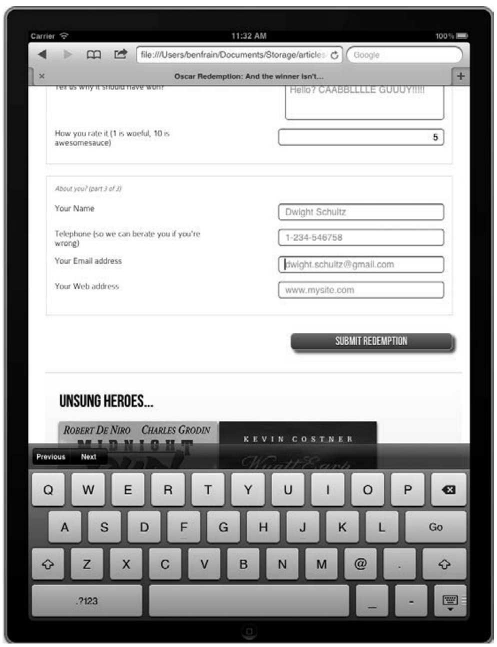
`type="email"`——支持它的浏览器会期望用户的输入匹配电子邮箱的格式。下面的示例代码中我们将 `type="email"`与`'required'`和`'placeholder'`组合起来使用：

```
<div>
  <label for="email">Your Email address</label>
  <input id="email" name="email" type="email"
    placeholder="dwight.schultz@gmail.com" required aria-required="true">
</div>
```

当提交一个不符合格式的输入值时，浏览器会生成警告信息：



此外，许多触摸屏设备（如 Android、iPhone 等等）会根据输入类型改变键盘模式。下图显示了 `type="email"`的输入框在 iPad 上的使用效果。注意看输入键盘上那个方便输入邮箱地址的@符号：

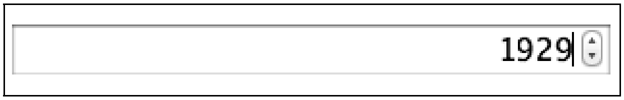


2. number

`type="number"`——支持该特性的浏览器期望输入一个数字。这种输入类型默认还提供控制按钮，允许用户简单地点击向上或向下来改变数值。代码示例如下：

```
<div>
  <label for="yearOfCrime">Year Of Crime</label>
  <input id="yearOfCrime" name="yearOfCrime" type="number" min="1929"
    max="2015" required aria-required="true" >
</div>
```

下图展示了该输入框在支持该特性的浏览器中的效果（Chrome v16）：



如果你输入一个非数字，浏览器会做些什么呢？Chrome（v16）会在输入框失去焦点时清除输入值，除此之外没有其他反馈；而 Firefox（v9）则允许你输入任何值（变成了一个标准的文本输入框）。你可能注意到在上面的代码中，我们还设置了允许输入的最小值和最大值范围，具体代码如下：

```
type="number" min="1929" max="2015"
```

超出范围的数字会（应该）得到特殊对待。浏览器的具体实现也不相同，Chrome（v16）会显示一个警告信息而 Firefox（v9）则什么都不做。

3. url

`type="url"`——你猜对了，URL 输入类型用于输入 URL 地址。与 `tel` 和 `email` 输入类型类似，它看起来和标准的文本输入框一样。不过有些浏览器会在提交不合法的 URL 时显示特定的警告信息。对应的代码示例如下，示例中包含了 `placeholder` 属性：

```
<div>
  <label for="web">Your Web address</label>
  <input id="web" name="web" type="url" placeholder="www.mysite.com">
</div>
```

下面的截图显示了在 Chrome 中提交一个不合法 URL 地址时的效果：



和 type="email"类型一样，触摸屏设备也会为 URL 输入框修改键盘模式。下图显示了 iPad 上 type="url"的使用效果：



看到键盘上的 Go、斜杠（/）和.com 这几个按键了吗？因为我们要输入 URL 地址，所以设备为我们专门显示了匹配的键盘（除非你没准备访问.com 网站，这种情况下你就用不着感谢苹果公司）。

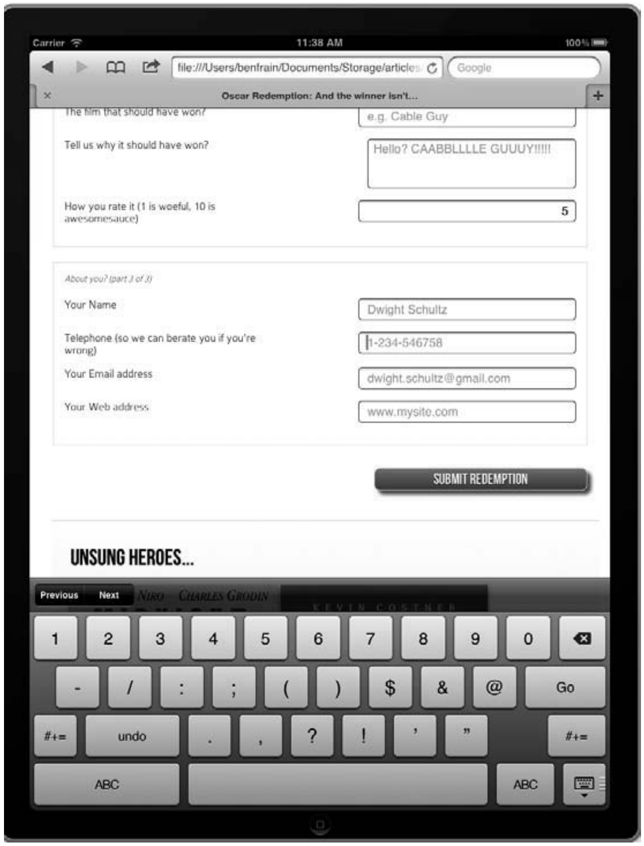
4. tel

type="tel"是另一种用于收集联系人信息的输入类型。tel 表示表单域期望输入一个电话号码。示例代码如下：


```
<div>
  <label for="tel">Telephone (so we can berate you if you're wrong)</label>
  <input id="tel" name="tel" type="tel" placeholder="1-234-546758"
    autocomplete="off" required aria-required="true" >
</div>
```

尽管 `tel` 类型被设计为输入数字格式，但在包括较新的 Chrome v16 和 Firefox v9 在内的很多浏览器中，它的表现和普通文本输入框一样。目前这些浏览器没有对无效输入值提供任何合理的警告信息[1]。

不过好的一点就是，和对待 `email` 和 `url` 类型一样，触摸屏设备为这种类型贴心地提供了数字键盘以便完成输入。`tel` 输入类型在 iPad 上的使用效果如下（iOS 5 上的运行效果）：



注意看键盘上是不是少了字母按键，而优先显示了数字按键？这样就可以让用户更快地输入正确的数值。

5. search

`type="search"`——和普通文本输入框的表现基本一样，仅在个别浏览器中渲染得有点细微差别。示例代码如下：

① 电话号码的输入格式和验证规则，不同地区可能有所不同。不提供统一验证也可以理解。——译者注

```
<div>
  <label for="search">Search the site...</label>
  <input id="search" name="search" type="search" placeholder="Wyatt Earp">
</div>
```

在 Firefox（v9）中的显示效果如下图，注意输入框的默认样式是矩形状：



但是在 Chrome（v16）中的默认渲染效果则稍有不同，输入框是圆角的，且输入框右侧有一个快速清除按钮：



6. pattern

`pattern=""`——“害怕，非常害怕”（还记得这句台词出自哪部电影吗？[①]）。在我看来，这句话用来描述正则表达式很合适。如果你不知道正则表达式是何物，那么我只好说不知是福。如果你知道，而且更厉害的是你还理解其含义，那下面的内容就是给你准备的。

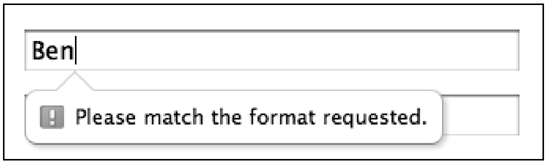
> **学习正则表达式**
>
> 如果你在万圣节的午夜一个人在墓地里看过《驱魔人》[②]，那你就可以去学习正则表达式了：http://en.wikipedia.org/wiki/Regular_expressions。

你可用 `pattern` 属性通过正则表达式来定义表单域的数据格式。示例代码如下：

```
<div>
  <label for="name">Your Name (first and last)</label>
  <input id="name" name="name" pattern="([a-zA-Z]{3,30}\s*)+[a-zA- Z]{3,30}"
        placeholder="Dwight Schultz" required aria- required="true" >
</div>
```

我在网上搜索了大约 458 秒终于找到了一个验证姓名的正则表达式，这是作者的义务。在 `pattern` 属性中放置一个正则表达式，支持该特性的浏览器就会按照指定格式验证输入值。当和 `required` 属性组合使用时，一旦输入不符合格式的值，浏览器就会有如下图所示的反应。本例中我的输入值缺少姓氏：

① 出自《the Fly》，中文名《变蝇人》。
②《驱魔人》http://movie.douban.com/subject/1293755/。

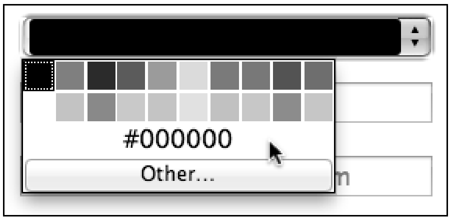


7. color

type="color"——会在支持该特性的浏览器中生成一个颜色选择器，让用户可以选择十六进制的颜色值。示例代码如下：

```
<div>
  <label for="color">Your favorite color</label>
  <input id="color" name="color" type="color">
</div>
```

悲剧的是，目前很少有浏览器支持该特性，好像只有 Opera（v11）支持。如果浏览器默认的颜色选择器中没有所需的颜色，点击下方的 Other...按钮就会打开操作系统默认的颜色选择器：



8.1.8　日期和时间输入类型

新的 date 和 time 输入类型背后的设计思想，是想为选择日期和时间提供一致的用户体验。如果你曾经在网上买过演出门票，那你就可能用过某种日期选择器。这种功能一般都是由 JavaScript（如 jQuery）提供，但我们希望能仅通过 HTML5 标签就实现这种常用功能。

1. date

示例代码如下：

```
<input id="date" type="date" name="date" />
```

和 color 类型一样，目前对 date 提供原生支持的浏览器寥寥无几，大多数浏览器默认都将其渲染为标准的文本输入框。超级棒的 Opera 已经实现了这个功能，下图显示了示例代码在 Opera（v11）中的渲染效果：



date 和 time 输入类型有很多不同的变种。下面是对其他变种类型的简要介绍。

2. month

示例代码如下：

```
<input id="month" type="month" name="month">
```

选择器界面允许用户选择某个月，输入框中会被填充为年和月组成的值，如 2012-06。

下图显示了浏览器中的效果：



3. week

示例代码如下：

```
<input id="week" type="week" name="week">
```

使用 `week` 类型时，选择器允许用户选择一年中的某一周，输入框中会被填充格式如 2012-W47 这样的数据。

下图显示了浏览器中的效果：



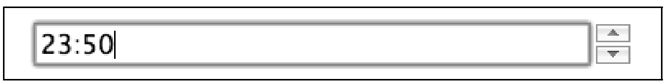
4. time

示例代码如下：

```
<input id="time" type="time" name="time">
```

`time` 输入类型允许输入一个 24 小时制的时间值，如 23:50。

在支持该特性的浏览器中，会显示出加减控制按钮，且仅允许输入时间值：

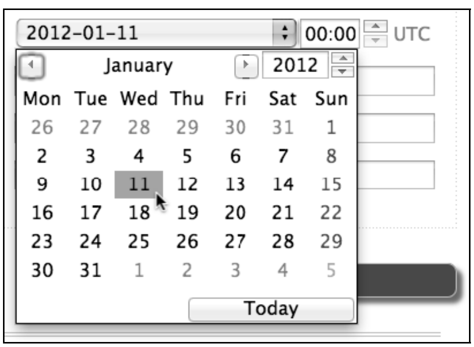


5. datetime 和 datetime-local

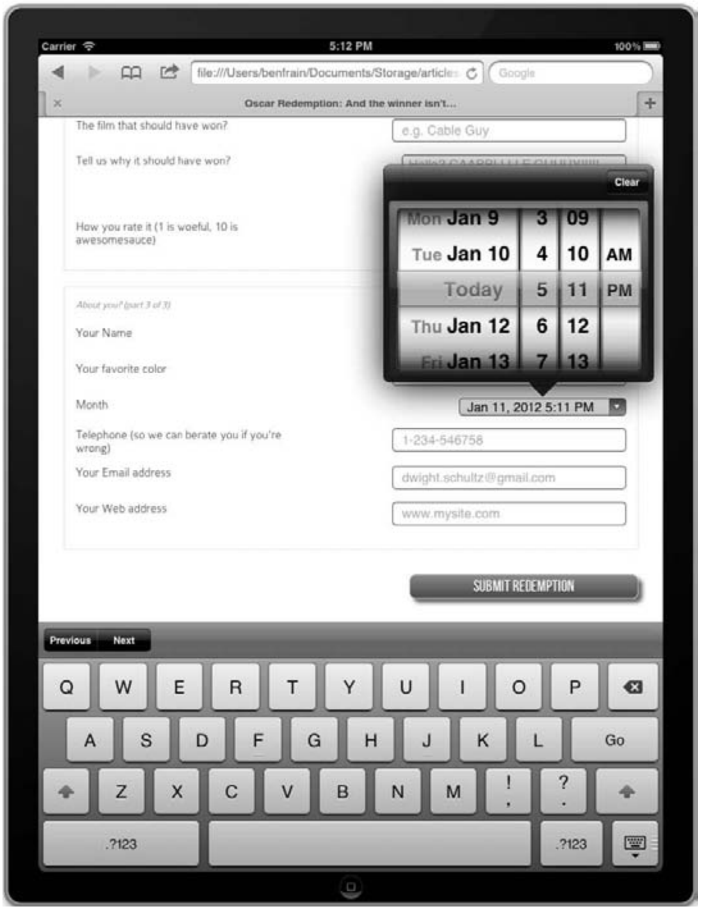
示例代码如下：

```
<input id="datetime" type="datetime" name="datetime">
```

在 Opera（v11）中的显示效果如下：



日期时间选择器在 iOS 设备上的显示效果更好，如下图：



日期时间选择器会生成一个由日期、时间和时区组成的值，日期与时间之间用 T 分割，使用 Z 表示协调世界时，使用正负值表示时区偏移。协调世界时的 2009 年 10 月 25 日显示如下：

```
2009-10-25T05:05:00Z
```

协调世界时（UTC）与格林尼治标准时（GMT）在大多数实际应用中基本一样。时区偏移则很好理解，比如，北京时间比格林尼治标准时提前 8 小时（UTC +8），反映到输入值上，效果如下：

```
2009-10-25T05:05:00+8:00
```

`datetime-local` 输入类型和 `datetime` 几乎完全一样，只是省略了时区信息。

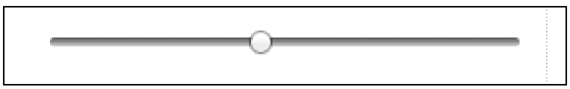
> **修改步增值**
>
> 你可以使用 `step` 属性来修改各种输入类型控制按钮的步增值。比如，要将时间的步增值改为 4 个小时，将 step 的属性值设为 14400 秒（4×60×60）即可。将 datetime 的步增值改为 4 小时的示例代码如下：
>
> ```
> <input id="datetime" type="datetime" name="datetime" step="14400">
> ```

6. range

`range` 输入类型会生成一个滑动条。示例代码如下：

```
<input id="howYouRateIt" name="howYouRateIt" type="range" min="1" max="10"
value="5" >
```

在 Safari（v5.1）中的效果如下图：



默认的输入范围是 0 到 100。但上面的示例代码通过 `min` 和 `max` 将范围设定为 1 到 10。

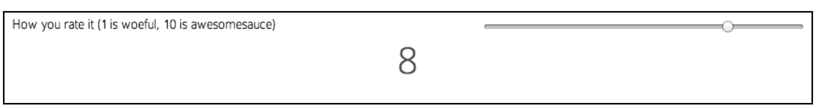
`range` 输入类型最大的一个问题是它从来不给用户显示当前的输入值。虽然滑动条仅被设计用来选择模糊的数值，但我还是经常想看看它的当前值。使用 HTML5 目前无法解决这个问题，但是如果你确实需要显示滑动条的当前输入值，可以通过 JavaScript 来实现。将上面的示例代码稍作修改：

```
<input id="howYouRateIt" name="howYouRateIt" type="range" min="1" max="10"
value="5" onchange="showValue(this.value)"><span id="range">5</span>
```

我们增加了两个东西，一个是 `onchange` 属性，另一个是 `id` 为 `range` 的 `span` 元素。接下来再将下面这段 JavaScript 代码加入页面：

```
<script>
  function showValue(newValue)
  {
    document.getElementById("range").innerHTML=newValue;
  }
</script>
```

这样做就能获取滑动条的当前输入值，将其显示在 `id` 为 `range` 的元素（`span` 标签）中。再用一点 CSS 将显示文字的字号变大、颜色变红即可。下图显示了修改后的滑动条在滑动过程中的显示效果：



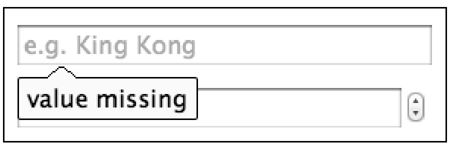
> 其实还有一些和表单有关的 HTML5 新特性，但是这些特性更多的都是和应用开发及服务器端开发相关，所以没在此处做讲解。想要参阅 W3C 有关 HTML5 表单部分的工作草案，请访问：http://dev.w3.org/html5/spec-author-view/forms.html#forms。

8.2　如何给不支持新特性的浏览器打补丁

前面把 HTML5 表单的功能吹上了天，但要想实际使用则还有两个非常麻烦的问题：一是支持表单新特性的浏览器在具体实现上有所不同；二是对完全不支持新特性的浏览器如何处理。万幸的是，Web 社区一如既往地拿出了解决办法。

回到第 4 章，我提过一个非常棒的 JavaScript 框架叫做 Modernizr（http://www.modernizr.com），用于向缺少 HTML5/CSS3 特性支持的浏览器打补丁。由 Alexander Farkas 编写的“Webshims Lib”（http://afarkas.github.com/webshim/demos/）就是构建于 Moderniz 和无处不在的 jQuery 之上的，它可用于插入表单补丁（也可以为其他 HTML5 特性打补丁），从而使不支持新特性的浏览器可以处理 HTML5 表单。最值得称道的一点是，它利用了 Modernizr 的加载功能，能做到只加载实际所需的补丁。如果在原生支持 HTML5 新特性的浏览器中查看网页，则仅会给网页加入一丁点儿冗余代码。而对于老版本浏览器，虽然它们需要加载更多的代码（因为它们本身能力不足），但通过相关 JavaScript 方法的辅助，它们能提供基本一致的用户体验。

通过打补丁受益的不仅仅只是老版本浏览器。我们知道，现在的很多浏览器也没有完全实现 HTML5 的表单特性。在网页中引入 Webshims Lib 可以弥补这些浏览器的缺陷。例如在 Safari 中，提交一个必填项为空的 HTML5 表单时不会有任何警告信息。其实这个表单根本不会提交，但它也没给用户任何反馈，这一点都不人性化。在页面中引入 Webshims Lib 后，上述问题会得到弥补：



当 Firefox（v9）无法给 `type="number"`属性提供控制按钮的时候，Webshims Lib 也可以提供一个合理的 jQuery 解决方案。总之，它是一个好用的工具，建议你立即下载这个小巧的工具包，然后在页面中使用，这样我们就可以继续编写 HTML5 表单，而所有用户都可以放心地看到他们需要的表单了（不包括使用 IE 6 且禁用 JavaScript 功能的那两个人——你知道我在说谁——快别这么干了！）。

首先下载 Webshims Lib（http://github.com/aFarkas/webshim/downloads）并解压。然后将其中的 js-webshim 文件夹复制到相应的位置。为简便起见，本例中我将其拷贝到网站的根目录，然后在页面的<head>部分加入如下代码：

```
<script src="js/jquery-1.7.1.js"></script>
<script src="js-webshim/minified/extras/modernizr- custom.js"></script>
<script src="js-webshim/minified/polyfiller.js"></script>
<script>
  //加载补丁
  $.webshims.polyfill();
</script>
```

分析一下这段代码。首先引入了一个本地的 jQuery 库文件（可以在 www.jquery.com 上下载最新版本）：

```
<script src="js/jquery-1.7.1.js"></script>
```

接着，又引入了 Webshims Lib 所包含的 Modernizr 以及补丁相关的 JavaScript 文件：

```
<script src="js-webshim/minified/extras/modernizr-custom.js"></script>
<script src="js-webshim/minified/polyfiller.js"></script>
```

最后，使用初始化脚本来加载所需的补丁：

```
<script>
  //加载补丁
  $.webshims.polyfill();
</script>
```

搞定。现在，浏览器缺失的新功能都会通过相关补丁脚本被自动追加进来。太棒了！

8.3　使用 CSS3 美化 HTML5 表单

现在我们的表单实现了跨浏览器一致的功能表现，同时也有一些基本样式。你我都知道，使用 CSS3 可以让表单的样式更美观。我们应用一些之前已经学习过的技巧，来使表单更加活色生香。目前我们的表单样式如下：

```
#redemption {
  width: 100%;
  font-family: 'ColaborateThinRegular';
  font-weight: 400;
}
#redemption hgroup {
  margin-bottom: 20px;
}
#redemption div {
  width: 100%;
  margin-bottom: 15px;
  float: left;
}
```

```
#redemption span#range {
  float: left;
  font-size: 3em;
  width: 100%;
  color: red;
  clear: both;
  text-align: center;
}
#howYouRateThis,#yearOfCrime {
  text-align: right;
}
#redemption legend {
  font-style: italic;
  color: #434242;
  font-size: 0.8em;
  margin-bottom: 20px;
  float: left;
  width: 100%;
}
#redemption fieldset {
  border: 1px dotted #cccccc;
  padding: 2%;
  margin-bottom: 20px;
}
#redemption label {
  width: 40%;
  float: left;
}
#redemption input {
  height: 20px;
  font-size: 1em;
  width: 40%;
  float: right;
}
#redemption textarea {
  height: 60px;
  font-size: 1em;
  width: 40%;
  float: right;
}
#redemption input#submit {
  text-decoration: none;
  height: 34px;
  font: 1.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: white;
  float: right;
  margin-bottom: 10px;
  background: linear-gradient(top, rgb(241,92,96) 0%, rgb(17
    100%);
  margin-top: 10px;
  box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
  text-shadow: 0px 1px black;
```

8

```
  border: 1px solid #bfbfbf;
}
.polyfill-important .input-range,.polyfill-important .step-c
  float: right;
}
.polyfill-important .step-controls {
  margin-right: -20px!important;
}
```

唯一需要注意的地方是最后两个样式只在相应的补丁脚本运行时发挥作用。

首先，我想让每个 `fieldset` 都有一个好看的渐变背景，相互之间稍微空开一点。下面是针对 `fieldset` 修改后的 CSS 代码：

```
#redemption fieldset {
  border: 1px dotted #cccccc;
  padding: 2%;
  margin-bottom: 20px;
  background: #ffffff;
  background: linear-gradient(top, #ffffff 77%,#f2f2f2 100%);
  border-radius: 4px;
  box-shadow: 2px 2px 5px hsla(0, 0%, 16.6667%, 0.3);
}
```

除了圆角和背景渐变效果，我们唯一需要做的就是再加一点点阴影（`box-shadow`）效果。

和前面的很多例子一样，我在此处省略了各种 CSS3 声明的浏览器私有前缀（本例中有 `linear-gradient`、`border-radius` 和 `box-shadow`）。

修改之后在 Chrome 中的效果如下图所示：



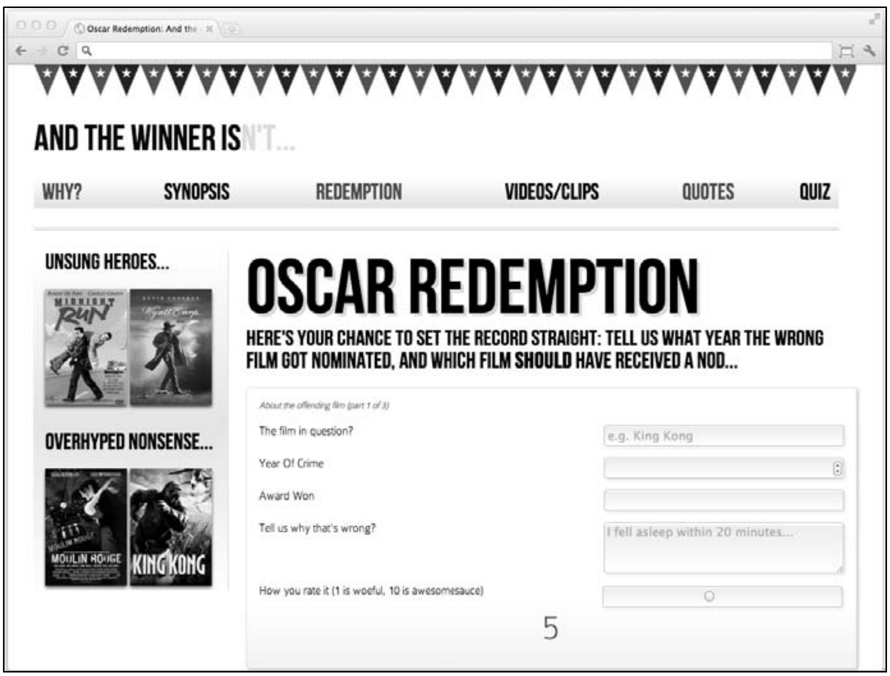
混用各种颜色值

在整个示例代码中，你可以看到我混用了各种颜色值。有时是 red 这样的颜色名称，有时是十六进制、RBG 或 HSL。在支持这些颜色模式的浏览器中这样做没什么问题。但在实际生产环境中，为保持一致性和兼容性，你应该只选用一两种颜色模式。

目前看起来效果还不错，但那些文本输入框依然有点枯燥。我们用下面的 CSS3 代码来为其润色一下：

```
input, textarea, select {
  border: 1px solid #bfbfbf;
  padding: 0.2em;
  font-size: 1.1em;
  line-height: 1.2em;
  background: #ffffff;
  background: linear-gradient(top, #ffffff 0%,#ededed 8%,#ffffff 100%);
  border-radius: 4px;
  box-shadow: 2px 2px 5px hsla(0, 0%, 16.6667%, 0.1);
}
```

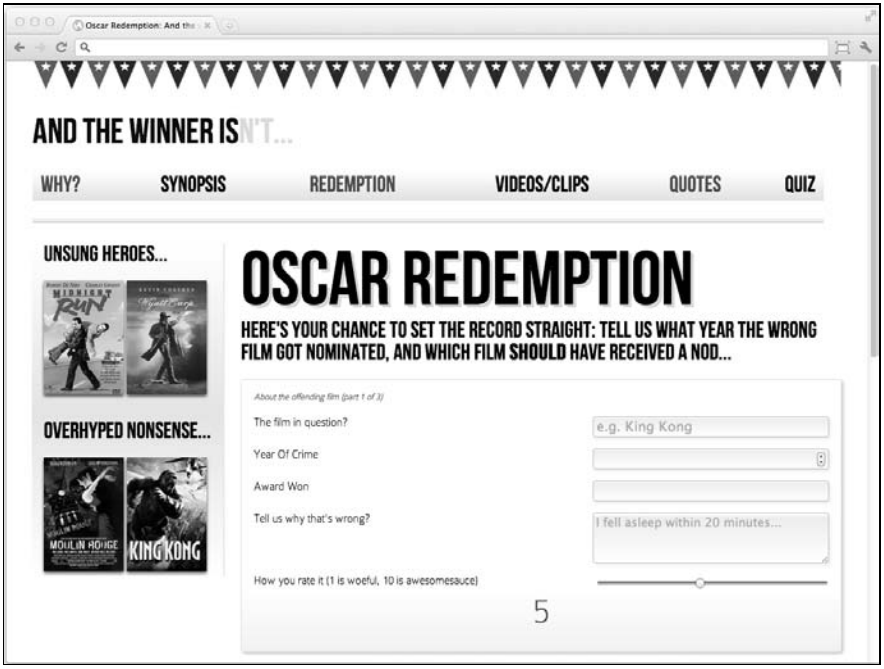
这里又一次使用了背景渐变，加了圆角，还有一点恰到好处的阴影。现在 Chrome 中的效果如下图：



这个效果我很满意……噢，等等。看看最下面的滑动条，这可不是我想要的效果。我不想让刚才的那些规则影响滑动条，所以我使用了一个 CSS3 选择器的新特性来选出我想要的元素：

```
input:not([type="range"]), textarea, select{
  /*样式*/
}
```

我使用`:not`这个否定选择器来排除`type="range"`类型的输入元素，再来看看 Chrome 中的效果：



太帅了！这就是我想要的效果，CSS3 不仅能轻松地追加样式，而且也能有效阻止在不需要的元素上追加样式。

针对表单的 CSS3 伪类选择器

除了前面已经了解的那些有趣的 CSS3 工具之外，CSS3 还提供了一些专门针对表单的伪类选择器。

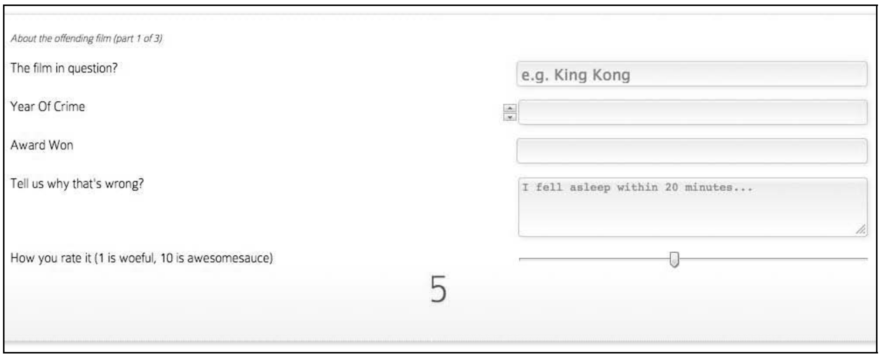
- `input:required`：选择必填表单域；
- `input:focus:invalid`：选择当前聚焦的且含有非法输入值的表单域；
- `input:focus:valid`：选择当前聚焦的且含有合法输入值的表单域。

我们使用上面的伪类选择器再编写三条新规则：

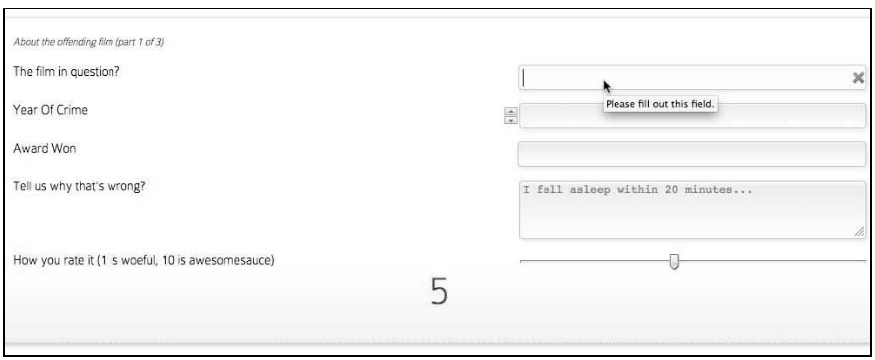
```
input:required {
  border: 1px solid rgba(253, 8, 8, 0.29);
}
input:focus:invalid {
  background: url('../img/cross.png') no-repeat right;
  padding-right: 3px;
}
input:focus:valid {
  background: url('../img/tick.png') no-repeat right;
  padding-right: 3px;
}
```

第一个规则给必填表单域加了一点红色边框。当用户输入时，如果所输入的值不符合规定格式，则第二个样式会给当前的输入框加上一个红色叉号；如果输入值符合格式，则第三个样式会给其加上一个绿色对号。

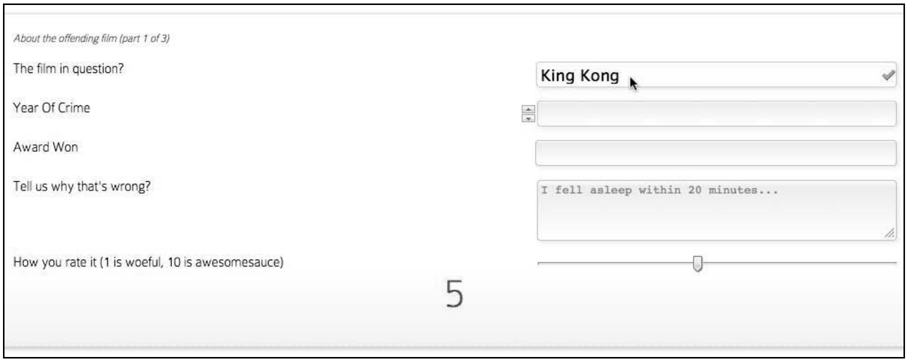
下面的截图显示了浏览器（Firefox v9）加载完页面的效果：



如果我们聚焦（点击）到一个必填输入框，右侧就会出现一个红色叉号（因为我们还没输入有效的值）：



如果我们输入一个有效的值，红色的叉号就变成了绿色的对号：



使用这些新的 CSS3 伪类选择器可以制作出优美的、易于实现的增强效果，可以大大提升用户填写表单的用户体验。

8.4　小结

在本章，我们学习了如何使用一堆新的 HTML5 表单属性。这些属性可以让表单比以前更好用，收集的信息比以前更精准。此外，我们可以通过 JavaScript 对象检测，有条件地加载 JavaScript 补丁脚本，从而保证这些新标签和属性是面向未来的，这样不论浏览器的功能如何都能为表单提供类似的用户体验。

我们的响应式 HTML5 和 CSS3 之旅就快结束了。我们讲述了大量理论，同时也制作了 AND THE WINNER ISN'T 这个示例网站。但是，在实际制作响应式网站时经常会遇到更多的挑战。如何处理小屏幕上的一大堆导航链接？如何仅为有实际需要的浏览器加载额外的资源文件？在最后一章中，我们将探讨基于 HTML5 和 CSS3 的响应式设计经常遇到的问题（以及解决方案）。我们还会再说说，如何以最佳方式处理那些有特定缺陷的老版本浏览器。

9 解决跨浏览器问题

在最后一章，我们将学习以下内容

- 渐进增强和优雅降级之间的根本区别
- 如何让老版本 Internet Explorer 支持响应式设计
- 何如使用 Modernizr 按需加载 CSS 文件
- 何如使用 Modernizr 按需加载 JavaScript 补丁文件
- 如何将超长导航列表在小视口中转换为选择菜单
- 如何为高分辨率（视网膜）显示屏提供匹配图片

在学习最后一章之前，先回顾一下我们当前的处境和状况。

移动设备使用量呈爆炸式增长。因此用户会使用各种各样的视口（不同的尺寸和方向）和带宽来浏览网页。在可预见的未来，我们需要以基本内容、分层特性和渐进增强为基准来设计和制作网站。此外，考虑到带宽差异，网站代码应该尽可能保持简洁和灵活。

从设计角度看，我们已经完整学习了 Ethan Marcotte 提出的响应式设计的三种方法。CSS3 媒体查询（第 2 章）用于创建设计断点，从而使网页布局可以完美匹配各种视口。而结合流动布局的弹性图片和多媒体技术（第 3 章），则可以保证网页在这些媒体查询断点之间的平滑过渡。响应式设计的结果就是设计不仅在今天流行的视口上表现完美，而且是面向未来的。

为了保持代码简洁，第 4 章我们将页面标签换成了 HTML5。HTML5 能提供更加精简和语义丰富的代码，而且为我们带来了诸如离线访问等新特性。此外，我们还在页面代码中使用了一些无障碍网页应用技术（WAI-ARIA）以增强可用性，为依赖屏幕阅读器和辅助技术的用户提供额外的帮助。

在第 5 章和第 6 章，我们领略了 CSS3 的强大威力和灵活性，学习了新的 RGBA 和 HSLA 颜色模式，以及如何不依赖图片而仅使用 CSS3 来制作盒阴影、文字阴影、背景渐变等各种常见的设计效果。此外，强大的 CSS3 选择器可以让我们选择 DOM 中任意需要的元素，而这种能力以往必须依靠 JavaScript 才能实现。但 CSS3 不只能用来调整设计效果或者大幅降低浏览网页所需的带宽消耗，更重要的是还为我们带来了一些在以往没有 Flash 或

JavaScript 就不敢奢望的新特性：自定义字体（第 5 章）以及不同视觉状态间的优雅平滑的过渡效果（第 7 章）。放眼未来，我们还了解了一些比较复杂的特性，如 CSS3 的 3D 转换。

最后在第 8 章，我们告别了单调乏味的表单制作体验，享受着使用 HTML5 标签来处理表单验证和创建表单元素的惬意。值得称道的是，我们还有条件地为一些老版本浏览器如 IE6、7 和 8 插入了 JavaScript 备用方案以提升用户体验。

从本书的开头到现在，我们用 HTML5 和 CSS3 完成了一个名为 And The Winner Isn't…. 的响应式网站。你可以通过 http://www.andthewinnerisnt.com 访问该网站。

下图展示了网站首页在 iPhone 上的显示效果：



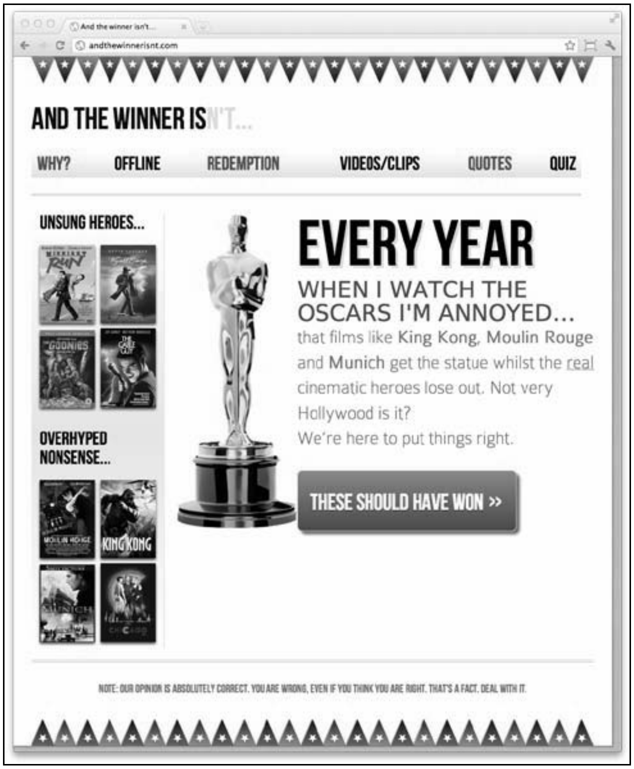
在 iPad 上的显示效果：



在 Android（模拟器）浏览器中的显示效果：



在现代桌面版浏览器（Google Chrome v16）中的显示效果：



以及在 Internet Explorer 8 中的效果：



我的妈呀！给我一把左轮手枪……

Internet Explorer 8 中的效果惨不忍睹，因为它不支持`<aside>`、`<header>`、`<nav>`和`<footer>`等 HTML5 元素，这自然就引入了本章的主题——解决跨浏览器问题。

9.1 渐进增强与优雅降级

你可能听过“渐进增强”和“优雅降级”这两个词。它们是处理多浏览器支持的两种方法论，并在 Web 社区中引发过激烈的辩论。虽然最初它们看起来是两个可互换的概念，但本质上则完全相反。下面是我的理解。

优雅降级指的是为现代浏览器制作网站，然后保证为某些老版本浏览器提供基本可用的体验。新特性在老版本浏览器中会降级，且一般会有一个分界点，声明不支持那些老掉牙的浏览器。有些时候用户也仅会被警告他们所使用的浏览器有问题，建议其更换（如“您的浏览器老得让人笑话——建议下载最新版浏览器!”）。

渐进增强与优雅降级恰好相反。渐进增强以恪守 Web 标准的标签为基础，意味着它在所有浏览器中均可用。然后通过 CSS 样式和必要的 JavaScript 来为更先进的浏览器提供渐进式的增强体验。

讨论过这两种方法优劣的文章数以百计。我们先来看看来自 Opera 开发者社区的这篇：http://dev.opera.com/articles/view/graceful-degradation-progressive-enhancement/，以及来自 Aaron Gustafson 的经典之作：http://www.alistapart.com/articles/understandingprogressiveenhancement。

现状

目前，渐进增强被普遍认为是开发网站的最佳实践。然而，冰冷的现实是，我虽然打心眼里喜欢渐进增强并用它来制作网站，但很多情况下可以说我是用优雅降级的方式来做的。怎么会这样呢?

www.andthewinnerisnt.com 这个网站是使用 HTML5 制作的。IE6、7、8 是在 HTML5 之前就已被开发和发布出来了（你应该知道，HTML5 虽然日益普及，但它还不是一个被正式批准的标准），所以无法识别`<aside>`、`<section>`和`<footer>`这样的标签。因此，从纯粹意义上讲我不应该使用 HTML5 元素。追加一段 JavaScript 来解决这个功能问题——这就是真正的渐进增强吗?

尽管如此，除非有什么让我信服的理由，否则我都一直选用 HTML5 来替代 HTML 4.01。现实是，根据我的日常工作经验，HTML5 利大于弊。在使用 HTML5 时，一定要编写与标准兼容的 HTML 代码（可以使用 HTML5 验证工具来排除代码错误，地址是：http://validator.nu/或 http://validator.w3.org/），好让所有设备都能以原生方式完美渲染。

无论如何，有时你肯定会选择（或是被逼无奈）去实现一些现代浏览器提供的增强功能，可能是为了那些老弱的 IE。比如你想让它能支持圆角，不过在你这么做之前，请再听我啰嗦几句……

9.2　该不该修复老版本 IE

我想再次重申以前的观点：通过打补丁的方式可以为老版本浏览器增加大多数 HTML5 和 CSS3 特性，但导致的结果是用户体验将严重依赖 JavaScript，且可用性相较于不使用补丁有所减弱。毫无疑问，影响这个抉择最关键的因素是性能。因为我只是告诉你能这么做，并不代表你应该这么做！

此外，即使不使用补丁（我们稍后会谈谈这个话题），以我的经验，为了能让 IE6 和 IE7（以及少量 IE8 和 IE9）的页面渲染效果尽可能与现代标准浏览器类似，开发、测试和配置专属 CSS 代码所花费的时间，至少也和为现代浏览器提供渐进增强所花费的时间一样多——这很让人不爽！你和你的客户就准备这样用掉开发时间？

9.2.1　统计数据（再看看世界的变化）

我们来重温一下第 1 章说过的话题。我们注意到从 2010 年 7 月到 2011 年 7 月，全球移动浏览器的使用率（数据来自 Global Stats 网站 http://gs.statcounter.com）不断上升，从 2.86% 涨至 7.02%，而 IE7 的使用率则继续下跌，低至 5.45%。2011 年最后一个月的统计数据则更加令人警醒：IE7 的使用率只剩了 4%，IE6 的使用率仅有 1.78%。而与此同时移动浏览器的使用率则增长至 8.04%。

另一个更加有趣的现象是截止 2011 年 12 月，Google Chrome（包括 v15 和 v16）仅凭一己之力就占据了全球浏览器市场的 25.7%，几乎与 IE 6、7、8 的总和（27.9%）持平。再加上其他现代浏览器的份额，如 Safari（4.3%，不包括 iOS 版）和所有版本的 Firefox（21.01%）以及各种移动浏览器，就不难发现为现代浏览器开发增强的用户体验，远比为那些老不死的玩意修补漏洞更有意义。至少对我来说是这样的！

结论就是：老弱的 IE（6、7、8）的使用率正在不断减少，而现代浏览器（包括桌面版和移动版）的使用率则正在不断增加。

9.2.2　个人选择

目前我个人的态度是，针对全新制作的网站，要确保其在最新版本的 IE（编写本书时最新版本是 IE9）以及前一个版本（IE8）中效果完美。而针对老版本 IE 的布局和样式调整则由于需要额外的时间，所以另行商议。

这样做并不表示我完全不管 IE7 等老版本浏览器的基本可用性问题。我只是将有限的开发时间用于保证基本布局和功能正常，所以会忽略一些微小的对齐问题，以及一些由于浏览器不支持背景渐变、圆角、盒阴影新特性而导致的视觉差异问题。这些东西不影响可用性，多半都属于渐进增强的效果，我也没期望在老弱浏览器上看到什么效果。

在多个浏览器中测试网站

通常，标准浏览器如 Chrome、Safari、Firefox 在渲染使用 HTML5 和 CSS3 制作的网页时，效果基本一致。目前，大多数智能手机使用的也是和桌面版 Safari、Chrome 一样的 Webkit 内核，所以其渲染效果也满足你的期望。但是，不同版本的 IE 渲染效果差别较大，所以毫无疑问你需要在各版本的 IE 中测试一下网站效果（除非你默认就使用 IE 浏览器——哎哟，我得向老同志表示一下慰问）。我通常都使用 IE Tester（http://www.my-debugbar.com/wiki/IETester/HomePage）这个免费的工具，利用它可以在一台机器上同时运行多个版本的 IE。不过还有很多类似的替代软件，Smashing Magazine 上的这篇文章对这类工具做了一个很好的总结：http://www.mashingmagazine.com/2011/08/07/a-dozen-cross-browser-testing-tools/。

为了说明这种方法，在看完 http://www.andthewinnerisnt.com 网站在 IE8 中的效果之后，很明显我们得做点什么，好让网站基本正常。我们准备使用一个名为 Modernizr 的 JavaScript 工具和一个补丁来修补老版本 IE 的缺陷。我不太确定 IE 是否值得这么做，因为毕竟问题是因它而起的，但谁让我心肠软呢。在我们开始之前，先来了解一下 Modernizr。

9.3 前端的瑞士军刀：Modernizr

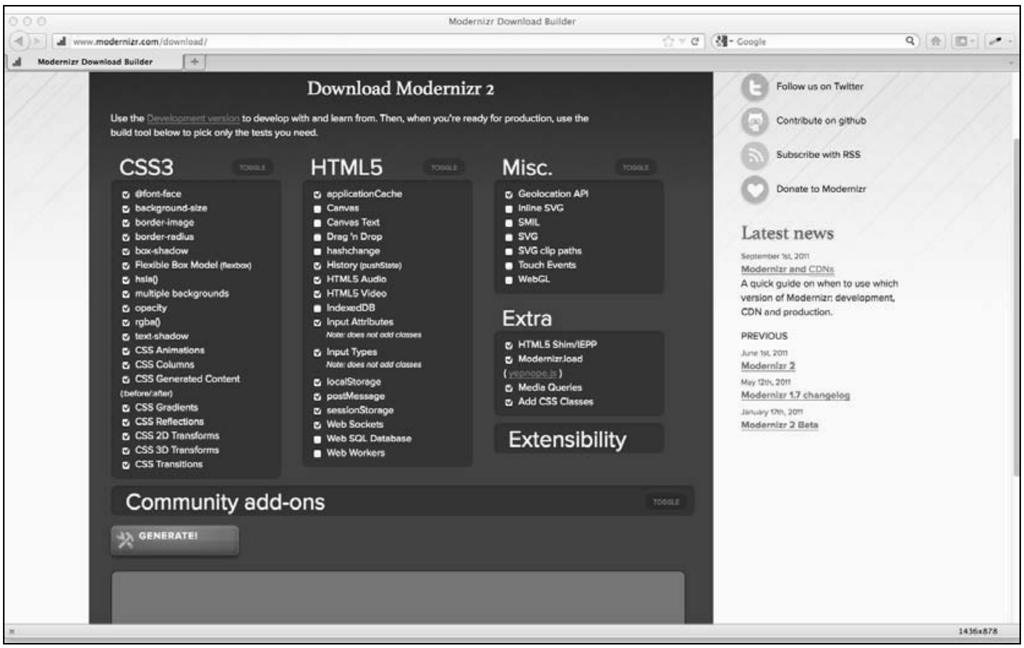
在线社区总能发现各种浏览器的兼容性问题，并不断为我等凡人贡献解决方案，这一点让我既赞叹又着迷。我们在第 4 章简要介绍过 Modernizr，到了最后一章又一次讲到它。Modernizr 是一个用于检测浏览器功能的开源 JavaScript 库。Modernizr 的第一版由 Fauk Ate? 开发，目前由 Alex Sexton 和才华横溢的 Paul Irish 担当首席开发人员。一些知名的公司已经开始使用这个工具，包括 Twitter、微软和谷歌。我说这些不只是为了吹捧 Modernizr 的开发团队（他们确实值得吹捧），更重要的是想说明 Modernizr 是 JavaScript 杰作，绝不会昙花一现。坦率地说，它是一个值得深入理解和学习的工具。

那它到底做了什么？既能给老版本浏览器打补丁，又能保证新浏览器渐进增强的用户体验，它是怎么做到的，而我们又如何使其听任差遣呢？好好听着……

从实际操作来看，Modernizr 默认做的事很少，除了（在你选择的情况下）给不支持 HTML5 标签的浏览器如 IE6、7、8 追加一点由 Remy Sharp 开发的 HTML5 垫片脚本，使其可以识别`<aside>`、`<section>`等 HTML5 元素之外，它主要做的就是浏览器“功能检测”。因此，它知道浏览器是否支持各种 HTML5 和 CSS3 特性。这就为我们根据浏览器特性采取不同行动提供了依据，剩下的事情就是我们如何具体实施了。所以，我们来给页面加入 Modernizr 来试试效果。

第一步，下载 Modernizr（http://www.modernizr.com）。

> **如何选择 Modernizr 版本——开发版还是生产版？**
>
> 如果你对 Modernizr 的运行原理感兴趣，那就下载开发版，因为每一个选项和功能测试都有相应文档。不过，开发版可以让你只选择仅和你的网站或 Web 应用相关的功能检测，这样可以保证文件既小巧又有的放矢。



接下来，将文件保存在某个适当位置（和之前一样，我保存在根目录下的 js 文件夹中），然后在页面的`<head>`中调用：

```
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-
    scale=1" />
<title>And the winner isn't…</title>
```

```
<link href="css/main.css" rel="stylesheet" />
<script src="js/modernizr.js"></script>
</head>
```

使用了 Modernizr 之后，再通过 Firebug 或类似开发工具查看页面源代码时，就会看到 HTML 标签上追加了一堆不同的类名。在 Firefox v9.01 中查看的示例效果如下：

```
<html class="js flexbox geolocation postmessage indexeddb history websockets rgba
hsla multiplebgs backgroundsize borderimage borderradius boxshadow textshadow
opacity cssanimations csscolumns cssgradients no-cssreflections csstransforms
no-csstransforms3d csstransitions fontface generatedcontent video audio
localstorage sessionstorage applicationcache" lang="en">
```

这种做法很棒。这样就可以区别具体的浏览器，告知我们它检测了那些特性，其中该浏览器支持哪些，不支持哪些（对不支持的特性，对应的类名前有 no-前缀）。基于此我们就可以做两件事：在 CSS 文件中逐个特性地修正样式，以及按需加载额外的 CSS 或 JS 文件。

9.3.1 使用Modernizr辅助修正样式问题

And the winner isn't…网站非常适合使用 Modernizr 来辅助修正一个样式问题。我们的问答页面在支持 3D 变形的浏览器中效果正常，在不支持该特性的浏览器中则仅做了一个简单的悬停效果。现在，无论浏览器是否支持 3D 变形，我们在页面上都有一段备注告知用户：该页面效果需要 3D 变形的支持。如果你没看到海报图片的 3D 翻转效果，试试 Safari 或 Chrome。

有了 Modernizr 追加的额外类名，就有办法仅在不支持 3D 变形特性的浏览器中显示备注信息了。

```
.note {
  display: none;
}
.no-csstransforms3d .note {
  display: block;
}
```

分析一下，首先将备注设置为默认不显示：

```
.note {
  display: none;
}
```

这意味着支持 3D 变形特性的浏览器（如 Google Chrome 16）就不会看到备注信息（如下图）：



接下来的规则使用了 Modernizr 追加的额外类名来为不支持 3D 变形的浏览器显示备注：

```
.no-csstransforms3d .note {
  display: block;
}
```

下图显示了同一页面在 Firefox 9 中的效果：



Modernizr 让我们抛弃浏览器的角度，转而从功能特性的角度考虑问题。

9.3.2　使用Modernizr让老版本IE支持HTML5 元素

我选择了一个含有 HTML5 垫片脚本的 Modernizr 自定义生产版，然后在 IE8 中刷新页面，可以看到页面效果比之前好太多了：



我不用再做什么。因为 Modernizr 使老版本 IE 可以识别 HTML5 的结构化元素，所以之前的很多标准 CSS 样式现在都可以识别了，页面呈现出了它应有的样子。

对于我这个自由职业者的钱包来说，这是天大的好事。如果你之前没有在现代浏览器中看过这个网站的效果，你应该都不知道两者有什么区别。不过，由于 IE8 缺少 CSS3 支持，所以相对于现代浏览器来说有一些明显的视觉效果上的缺陷。导航链接上没有交替颜色（如果需要，我们可以通过给奇数导航链接追加额外类名来解决这个问题），按钮没有圆角，没有文字阴影和盒阴影。最重要的一点是，虽然我们的流式布局很灵活，但缺少 CSS3 意味着不支持媒体查询。没有媒体查询，在 IE 6、7、8 中就不会有不同视口中精心设计的布局变化。

虽然我怎么都不认为 IE8 中这样的布局“毁”了设计效果，但只要你愿意，Modernizr 也可以帮你修补老版本浏览器。为了证明这一点，我们来给 IE 6、7、8 追加 `min/max-width` 媒体查询支持，从而使网页在这些浏览器的不同视口中也能正确响应。

9.3.3 给IE6、7、8 追加min/max媒体查询功能

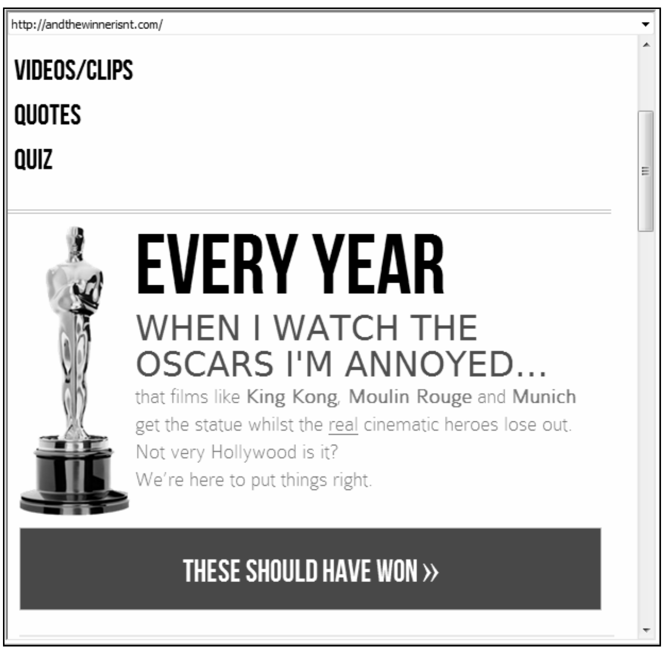
我选择的给老版本 IE 添加媒体查询支持的垫片脚本，仅支持追加 `min/max-width` 媒体查询功能。还有很多类似的垫片脚本，可以追加很多媒体查询特性。但对于响应式设计来说，Scott Jehl 开发的 Respond.js 使用简单、速度快，我一直用得很顺手。

Respond.js（https://github.com/scottjehl/Respond）可以独立于 Modernizr 使用——只需将其引入到需要的页面，然后就如其作者 Scott Jehl 所说："撬开 IE，击掌相庆。"

在将 Respond.js 整合到 Modernizr 之前，我们先来单独使用一下。将 Respond.js 直接丢到页面上（即将其插入到 Modernizr 文件之后），然后检查看看它对 IE 做了哪些处理。具体做法是将 Respond.js 保存到某个适当的路径，然后在`<head>`中引用：

```
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-
    scale=1" />
<title>And the winner isn't…</title>
<link href="css/main.css" rel="stylesheet" />
<script src="js/modernizr.js"></script>
<script src="js/respond.min.js"></script>
</head>
```

现在，只要在 IE8 中加载页面并调整浏览器窗口，就会看到我们的响应设计效果又回来了（如下图所示）：



不错，我们现在给 IE 添加了垫片脚本，使其可以支持 `min-width` 和 `max-width`，但有一个问题：垫片脚本在每一种浏览器中都会被加载——不论它们需要与否。一种解决办法就是将脚本文件链接放置在 IE 条件注释中，像下面这样：

```
<!--[if lte IE 8]>
        <script src="js/respond.min.js"/></script>
<![endif]-->
```

你以前肯定见过条件注释。这是一种非常简便的方法，可用来为相应版本的 IE 加载 CSS 或 JS 文件（甚至是内容）。其他浏览器会将这些代码看做注释而直接忽略。

在本例中，我们的条件注释是说：“如果你的浏览器版本低于或等于（`lte`）IE8，就执行其中的代码。”

详细了解条件注释

相较于功能检测来说，条件注释并不太受欢迎，但如果你想了解更多有关条件注释的知识，请参阅如下网址：http://msdn.microsoft.com/en-us/library/ms537512%28v=vs.85%29.aspx

这样问题就解决了。但我们真要使用 IE 特有的条件注释来污染我们的标签代码吗？那针对其他浏览器的垫片脚本又该怎么办呢？还是得用 Modernizr 呀。

9.3.4 使用Modernizr按需加载资源

Modernizr 在保证网站或 Web 应用的代码精简方面有一个极大的优势，就是它能按需加载资源（CSS 和 JS 文件）。其实我们只需要加载用户真正需要的资源，而不是采用“贪大求全”的策略（不管他们是不是真正需要就）加载每一个用户可能用到的资源。这样就可以保证对每一个用户，我们的页面都足够简洁，加载足够快速。

之前 Modernizr 已经被追加到了页面头部，接下来我们来为那些不支持 CSS3 媒体查询的浏览器（如 IE6、7、8）加载 Respond.js 文件。

Modernizr 中包含一个名为 YepNope.js 的小型 JavaScript 库文件（http://yepnopejs.com/）。使用方法很简单：

```
Modernizr.load({
    test: Modernizr.mq('only all'),
    nope: 'js/respond.min.js'
});
```

首先调用 Modernizr 的资源加载方法：

```
Modernizr.load({
```

加载方法中包含功能检测，以及根据检测结果将要采取的一系列动作。在本例中，我们

检测的是浏览器是否支持媒体查询：

```
test: Modernizr.mq('only all'),
```

如果不支持，则会加载 respond.min.js 这个文件：

```
  nope: 'js/respond.min.js'
```

此处的 only all 表示“你能识别媒体查询吗？”，老版本 IE 的检测结果肯定是不支持，所以对应 nope 中的动作，即加载相应的资源文件。这样就可以让 respond.min.js 仅在需要的时候加载。

还可以一次加载多个文件：

```
Modernizr.load({
    test: Modernizr.mq('only all'),
    nope: ['js/respond.min.js', 'css/extra.css']
});
```

上面的代码中使用了一个数组来同时加载 respond.min.js 和一个名为 extra.css 的 CSS 文件。这种方法可以让你根据实际的需要来加载独立的样式。还有一点值得一提，即还可以根据不同的检测结果来加载不同的资源文件：

```
Modernizr.load({
    test: Modernizr.mq('only all'),
    yep: 'js/pass.js',
    nope: ['js/respond.min.js', 'fail-polyfill.js', 'fail.css'] ,
    both: 'js/for-all.js'
});
```

上面的代码在检测通过时加载一个文件，检测失败时加载三个文件，最后无论检测结果如何，都加载一个名为 for-all.js 的文件。

按需加载资源的代码可以单独写在一个 JavaScript 文件里。本例中我将其命名为 conditional.js，并将其与 modernizr.js 和 respond.min.js 一起放在 js 文件夹。这样整理之后，<head>部分看起来就是下面这个样子：

```
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initial-
scale=1.0,maximum-scale=1" />
<title>And the winner isn't…</title>
<link href="css/main.css" rel="stylesheet" />
<script src="js/modernizr.js"></script>
<script src="js/conditional.js"></script>
</head>
```

注意我已经移除了 respond.min.js，因为现在该文件可以根据需要自动加载。

> 有关使用 Modernizr 按需加载资源的更详细文档，请见此处：http://www.modernizr.com/docs/#load。

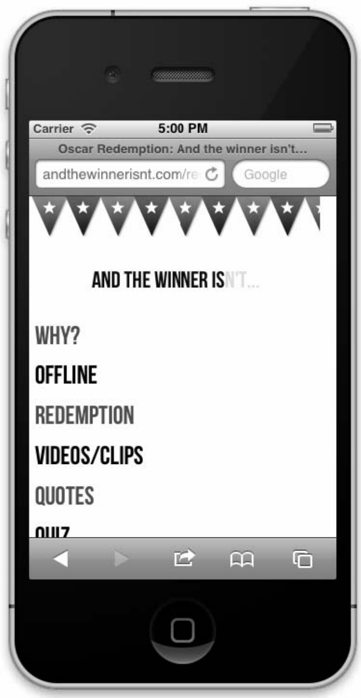
下载垫片脚本

下面的这个 github 库中有很多有用的垫片脚本：https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills。

9.4　必要时将导航链接转换为下拉菜单

在小视口屏幕中，响应式设计普遍存在一个问题，即页面上一大堆导航链接会占据宝贵的屏幕空间。

例如，And the winner isn't…网站只有 6 个导航链接，目前页面在小视口中的效果如下所示：



我想在浏览器视口小于一定宽度时，将这些导航链接转换为一个下拉菜单。你可以自己编写一段代码来做这件事。可敬的 Chris Coyier 写过一篇文章来讲怎么做（http://css-tricks.com/convert-menu-to-dropdown/）。不过，你还可以选择别人已经写好的脚本。为了简便起见，我就选了个现成的脚本。下图显示了在小视口中将导航链接转换为下拉菜单后的效果：



点击“Select a page”按钮就会显示出导航，效果如下：



这就是渐进增强的典型代表——它不是一个必需的功能，但为使用小屏幕设备的用户提供了更多的第一屏内容。那我们来继续看看如何实现它。首先下载 Responsive Menu 脚本（https://github.com/mattkersley/Responsive-Menu），和以前一样将其（`jquery.mobilemenu.js`）存放在 js 文件夹。接下来再给每个页面的导航链接部分设置一个 `id`：

```
<nav role="navigation">
  <ul id="mainNav">
    <li><a href="why.html">Why?</a></li>
    <li><a href="offline.html">Offline</a></li>
    <li><a href="redemption.html">Redemption</a></li>
    <li><a href="video.html">Videos/clips</a></li>
    <li><a href="quotes.html">Quotes</a></li>
    <li><a href="3Dquiz.html">Quiz</a></li>
  </ul>
</nav>
```

不做上面这一步也行，但 jQuery 选择器搭配特定 `id` 速度比较快。

接下来，在 `conditional.js` 文件中追加如下代码：

```
Modernizr.load([
  {
    test: Modernizr.mq('only all'),
    nope: 'js/respond.min.js'
  },
  {
    // 如果max-width等于600px，则加载菜单转换脚本
    test: Modernizr.mq('only screen and (max-width: 600px)'),
    yep : ['js/jquery-1.7.1.js', 'js/jquery.mobilemenu.js'],
    complete : function () {
      // 资源完全加载后运行
      $(document).ready(function(){

        $('#mainNav').mobileMenu({
          switchWidth: 600,                   //转换宽度（用px表示）
          topOptionText: 'Select a page',     //第一个选项的内容
          indentString: '   '  //缩进嵌套选项的字符串
        });
      });

    }
  }
]);
```

在为老版本 IE 按需加载 `Respond.js` 之后，我们增加了另一个检测：

```
test: Modernizr.mq('only screen and (max-width: 600px)'),
```

这个检测是问浏览器视口是否能识别媒体查询，如果可以，那视口的最大宽度是否是 600 像素？如果是，则：

```
yep : ['js/jquery-1.7.1.js', 'js/jquery.mobilemenu.js']
```

上面这行代码同时加载了 jQuery 库文件和 Responsive Menu 文件，

```
complete : function () {
……其他代码……
```

complete 这块代码意思是，一旦文件下载并执行之后，运行如下代码：

```
$(document).ready(function(){

        $('#mainNav').mobileMenu({
          switchWidth: 600,                    //转换宽度 (用 px 表示)
          topOptionText: 'Select a page',      //第一个选项的内容
          indentString: '   '  //缩进嵌套选项的字符串
        });
});
```

这里有几个针对 Responsive Menu 的配置参数。最重要的第一个参数，定义了在视口多宽的情况下，将导航链接转换为下拉菜单（此处我设置为 600px）。

此处我们使用 Modernizr 又一次保证了不需要该效果的用户不会加载多余代码，而需要的用户则会享受到这种渐进增强体验。

对于网页设计师，尤其是那些不熟悉 JavaScript 代码的人来说，一看到 Modernizr 就会心生畏惧。Modernizr 当然值得深入挖掘，但这个简单的教程已经足够说明 Modernizr 的用处了，它在你将来的响应式项目中一定会大显身手。

9.5　高分辨率设备（未来趋势）

设备及其性能总是与时俱进。事实上，我们不仅仅要面对不同的视口尺寸，还得考虑高分辨率显示屏。iPhone 4 是第一款被广泛使用的**高分辨率**设备。它的屏幕大小为 960 乘以 640 像素，分辨率为每英寸 326 个像素，是上一个版本 iPhone（iPhone 3GS）及 2011 年的 15" MacBook Pro 笔记本分辨率的两倍。预计其他设备，从平板电脑到笔记本电脑再到台式机都会跟风。幸好，我们的响应式工具已经提供了方法，可以为这类设备提供增强支持。

假设我们要为使用高分辨率设备的用户加载一个高分辨率版本的网站标志图片。最近在重新设计个人网站 http://www.benfrain.com 时，我就遇到了这种情况。我的网站标志区域的代码如下：

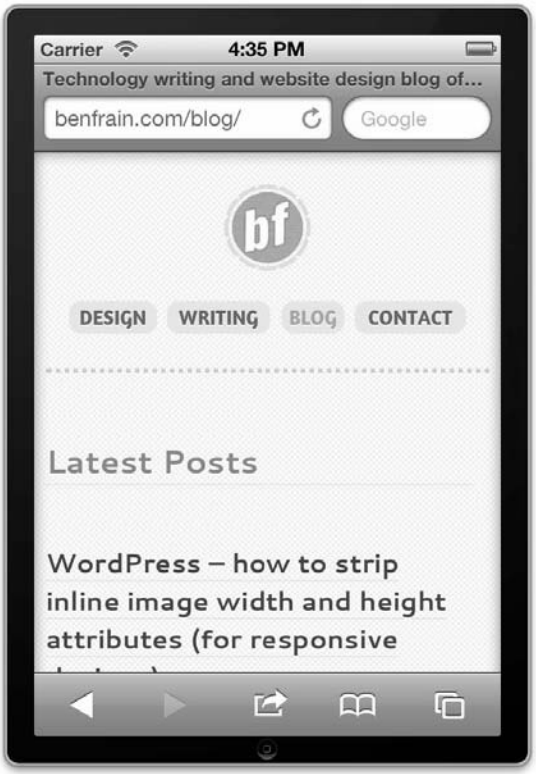
```
<div class="logo">
  <a href="http://benfrain.com/"></a>
</div>
```

对应的 CSS 规则如下：

```
 #container header[role="banner"] .logo a {
    background-image: url("../img/logo2.png");
```

```
    background-repeat: no-repeat;
    background-size: contain;
    display: block;
    height: 7em;
    margin-top: 10px;
}
```

最初显示效果如下：

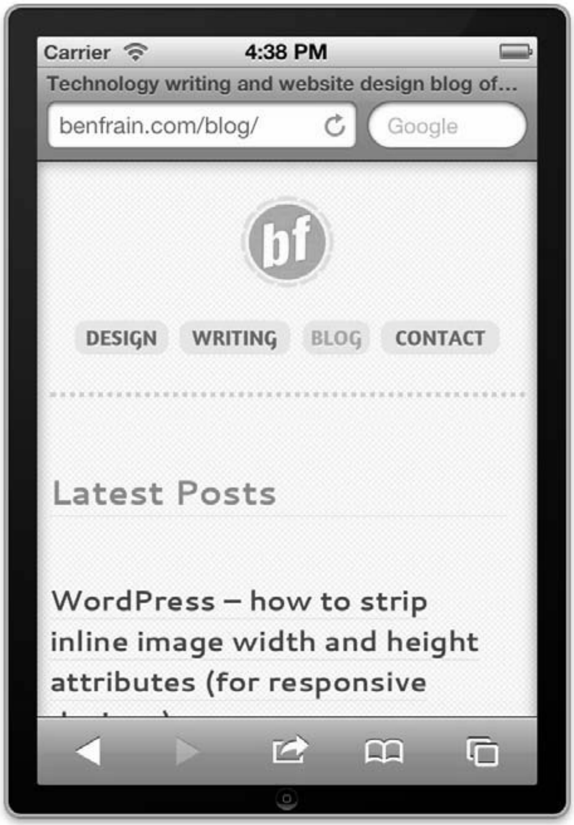


看起来很好，但我想让标志在高分辨率设备上也尽可能清晰。所以，我制作了两个新版本的图片（一个用于默认状态，一个用于悬停状态），图片尺寸是原来的两倍，然后将其分别命名为 `logo2@x2.png` 和 `logo2Over@x2.png`。之后在 CSS 中追加如下代码：

```
@media all and (-webkit-min-device-pixel-ratio : 1.5) {
  #container header[role="banner"] .logo a {
    background-image: url("../img/logo2@x2.png");
  }
  #container header[role="banner"] .logo a:hover {
    background-image: url("../img/logo2Over@x2.png");
  }
}
```

上面的媒体查询匹配最小设备像素比为 1.5 的设备。因此，像 iPhone 4 以及后续出现的高分辨率设备就会使用该媒体查询内的样式。你可能注意到样式中包含了`-webkit-`前缀。和以前一样，记得针对各种设备添加对应的私有前缀。

现在，在高分辨率设备上就会显示对应的高分辨率标志图片，效果如下：



无法否认，修改前后的差别微不足道。你最好还是使用真机亲身体验一下那种微妙的差别，图片的细节越丰富，在高分辨率屏幕上显示的效果也就越清晰。

使用这个技巧时需要慎重。图片尺寸越大，文件体积也越大，所需的加载时间会更长，所以再重申一遍：你可以这么做，但不一定应该这么做。

另外，如果浏览器支持**可缩放矢量图形**（SVG），那使用该技术替代图片可以解决我们目前面临的很多图片缩放问题。顾名思义，可缩放矢量图形能生成可任意缩放但显示清晰的矢量图片。不过，媒体查询和 SVG 对嵌入在页面中的高分辨率图片无能为力。在这种情况下你需要考虑基于 JavaScript 的解决方案。

9.6 小结

本章，我们分析了渐进增强与优雅降级的本质区别。随后使用垫片脚本让老版本 IE 可以识别媒体查询，从而使我们的设计在这些老弱浏览器中也能响应。接着，我们学习了使用 Modernizr 根据一系列功能检测按需加载 CSS 和 JavaScript 文件，从而为缺少某个特性的浏览器提供垫片脚本，以及额外或独立的样式。最后，我们还简单介绍了在不久的将来会极为常用的 CSS3 技巧，为相应的设备提供更好的增强体验。

此时此刻，关于制作响应式网站或 Web 应用所需的技术和工具，相信我都已经讲到了（但愿如此）。

我坚信，对目前的大多数网站来说，采用 HTML5 和 CSS3 的响应式网站设计，绝对是最佳方案。只要对现有的工作流程、实际做法和开发技巧稍作调整，就可以制作出快速、灵活、易维护的网站，最重要的是，无论在何种视口下网站都能表现完美。

随着移动设备的日益普及，我们之前从未预见到的新设备也将陆续加入上网冲浪的队伍。响应式设计无疑为我们提供了一套可靠且面向未来的方法，让我们的响应式网站无论在任何设备、任何视口中，也无论是否在线，都能快速响应。

“看完这本书，我必须得说，这时间花得太值了。作者通过一个虚构的网站，让我们掌握所有细节。”

“响应式Web设计具有巨大的潜在商业价值，这本书把响应式Web设计讲得非常透彻。”

——亚马逊读者评论

随着iPad mini的发布，又一个新的屏幕尺寸诞生了。用不着全面统计，你就会发现移动互联网时代众多的屏幕规格，从智能手机的3、4、5英寸，到平板电脑的7、8、9、10英寸，再到笔记本和台式机的13至30英寸，绝非目前单一的固定或流式布局所能应付。于是，响应式设计应运而生，而且它也将成为移动互联网时代前端设计与开发人员的一门必修课。

本书堪称学习响应式Web设计的难得佳作。它不仅全面、细致、图文并茂地介绍了响应式设计相关的技术，比如媒体查询、流式布局、弹性媒体和弹性字体等，还把近几年来Web设计领域公认的最佳设计理念有机地融入到了实例当中，比如移动先行（Mobile First）、渐进增强、平稳退化、无障碍设计等。更加难得的是，本书以设计跨屏幕的网页（响应式设计）为出发点，以点带面，把如今Web设计领域两大标准的最新版本HTML5和CSS3也纳入其中，读者在掌握先进设计方法的同时也能掌握最新的设计技术（比如使用新的HTML5结构化语义标记、嵌入媒体、响应式视频，以及CSS3的新选择器、特效、过渡、变形和动画等），从而可以免除重复学习新标准之苦，让自己一步跨入Web设计领域的最前沿。无论你想学习响应式Web设计，还是学习HTML5和CSS3的实际应用，本书都能满足你的需要，是毋庸置疑的明智之选。

说到底，响应式Web设计并非一门独立的技术，而只是现有技术的一个组合应用。只要有一点HTML和CSS基础的读者都能顺利地掌握它。对于中、高级的前端设计和开发人员，翻阅本书也有助于理清自己的知识脉络，对这个新的设计理念获得更全面、深入的理解和把握。

习惯移动阅读的读者，可访问图灵社区，购买本书电子版：http://www.ituring.com.cn/book/1055。



Responsive Web Design with HTML5 and CSS3

响应式Web设计
HTML5和CSS3实战



Ben Frain是一名具有十多年经验的网页设计师和前端工程师，直接与世界各地的客户和设计机构并肩工作。同时他还是一名技术记者，定期为一些关注Mac平台、前沿科技、网页设计和航空技术的刊物撰稿。

在此之前，他曾是一名怀才不遇的（而且谦虚谨慎的）电视演员，毕业于索尔福德大学的媒体与表演专业。他写了四部（自认为）同样被低估的剧本，而且始终心怀能卖出一部的信念（尽管不像最初那么强烈了）。

工作之余，在身体（和妻子）允许的情况下，他喜欢玩室内足球。 他的个人网站是www.benfrain.com，Twitter地址是twitter.com/benfrain。

PACKT PUBLISHING

图灵社区：www.ituring.com.cn
新浪微博：@图灵教育　@图灵社区
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议　计算机/Web开发与设计

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-29922-2

9 787115 299222 >

ISBN 978-7-115-29922-2
定价：49.00元